Carl von Ossietzky
Universität
Oldenburg

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Flexible komponentenbasierte Modellierung und Simulation von Szenarien für die Verifizierung und Validierung automatisierter maritimer Fahrsysteme

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften
der Carl von Ossietzky Universität Oldenburg
zur Erlangung des Grades und Titels

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

angenommene Dissertation

von Herrn **David Reiher**
geboren am 8. Juni 1990 in Stralsund

GutachterInnen:

Prof. Dr.-Ing. Axel Hahn
Prof. Dr.-Ing. habil. Andreas Rauh

Tag der Disputation:

30. April 2026

DANKSAGUNG

Geschafft! Der langjährige Weg zur Fertigstellung der vorliegenden Dissertation war gesäumt von vielen Steinen unterschiedlicher Größe. Einige waren intrinsischer Natur, andere wiederum wurden dort extrinsisch platziert. Manche haben lediglich viel Zeit gekostet, wie u. a. die vollständige Umstellung auf Online-Lehre während der COVID-19-Pandemie und der damit einhergehenden Anpassung aller Lehrinhalte, andere waren stark demoralisierender Natur und haben es mir, in einem mir bisher nicht bekannten Maße erschwert, an mich selbst und meine Fähigkeiten zu glauben. Dafür, dass ich nun trotzdem diese Danksagung für die zu veröffentlichende Version schreiben darf, will ich mir an dieser Stelle aber nicht selbst auf die Schulter klopfen, sondern denjenigen Menschen meinen Dank aussprechen, die auf unterschiedliche Weise dazu beigetragen haben, dass aus einer Idee, einigen Umwegen und einer beträchtlichen Menge Notizen schließlich diese Arbeit werden konnte.

Dank gilt zunächst Axel Hahn für die Ermöglichung der Bearbeitung dieses spannenden und praxisnahen Themas, das es mir ermöglicht hat, auch die in meiner vorangegangenen Zeit in der freien Wirtschaft gewonnenen Kenntnisse und Fähigkeiten einzubringen. Andreas Rauh danke ich für die Übernahme des Zweitgutachtens. Marco Grawunder danke ich für sein Einbringen als Mitglied der Prüfungskommission. Jürgen Sauer danke ich für die Übernahme des Vorsitzes der Prüfungskommission und die sehr angenehme berufliche Zusammenarbeit. Jürgen war nicht nur Leiter der Abteilung Systemanalyse und -optimierung, sondern auch stets der Ruhepol dieser, der einem mit Rat zur Seite stand oder mit einem großen Fundus an Geschichten aus den Anfängen der Apple-Computer für Auflockerung sorgte. Auch für das Vertrauen in Bezug auf die Gestaltung von Lehrinhalten möchte ich mich herzlich bedanken. Dein bevorstehender Abschied in den verdienten Ruhestand wird eine nur sehr schwer zu füllende Lücke an der Universität hinterlassen.

Den restlichen KollegInnen der Abteilung Systemanalyse und -optimierung der Universität Oldenburg gebührt ebenso mein Dank. Hervorzuheben ist dabei Manuela Wüstefeld: Sie ist die gute Seele der Abteilung und ist stets auf das Wohl ihrer KollegInnen bedacht. Einen besonderen Stellenwert hatten während meiner Zeit an der Universität Oldenburg auch die KollegInnen des An-Instituts OFFIS und des angrenzenden Instituts für Systems Engineering für zukünftige Mobilität des DLR. Die unzähligen Gespräche und Kaffeepausen haben sowohl fachlich als auch qualitativ zu dieser Arbeit beigetragen und für kurze Verschnaufpausen im akademischen Alltag gesorgt.

Meiner Mutter Britt und meinem Vater Thomas spreche ich besonderen Dank aus, denn sie haben mich auf meinem Lebensweg stets ermutigt, bestärkt, unterstützt und an mich geglaubt. Ohne diesen bedingungslosen Rückhalt hätte ich es niemals an den Punkt geschafft, an dem ich jetzt stehe.

Mein größter Dank gilt aber ohne Zweifel meiner Partnerin Jana. Du hast die guten Phasen mit mir gefeiert, die schwierigen ausgehalten und vermutlich mehr über Szenarien und Simulationen gehört, als ein Mensch freiwillig wissen möchte. Danke für deine Geduld, deinen unerschütterlichen Glauben an mich, dafür, dass du mir vor allem in den letzten Monaten ungefragt den Rücken vollständig freigehalten hast und dafür, dass du mich immer wieder daran erinnert hast, dass diese Arbeit wichtig ist, aber nicht alles. Fast genauso wichtig war der Beitrag meines Hundes Jabba, der gespürt hat, wenn es in mir mal nicht so gut aussah, und mir dann durch seine bedingungslose Zuneigung viel Stress abnahm. Ihr beide zusammen habt den größten Anteil daran, dass ich den Weg zur Promotion bis zum Ende gehen konnte. Ich liebe euch!

KURZFASSUNG

Für die flächendeckende Einführung hochautomatisierter Schiffe ist die öffentliche Akzeptanz dieser obligatorisch. Das dafür nötige öffentliche Vertrauen in die entsprechenden Technologien kann nur über die Sicherstellung der Funktionsfähigkeit und Sicherheit vollständig erreicht werden. Moderne softwarebasierte Fahrfunktionen, drastisch zunehmende Komplexität und die besonderen Merkmale der Entwicklungsprozesse moderner Schiffe stellen den Beweis der Sicherheit vor neue Herausforderungen, die es zu bewältigen gilt. Ein vielversprechender Ansatz ist das szenariobasierte Testen. Dabei werden Computersimulationen als die praktikabelste Möglichkeit zur Durchführung von entsprechenden Testreihen angesehen. Bestehende Simulationsanwendungen stoßen hierbei jedoch an ihre Grenzen und es mangelt an szenariobasiert nutzbaren Simulationsanwendungen, die flexibel als Testwerkzeug in vielfältige Entwicklungs- und Testprozesse integriert werden können.

Diese Arbeit befasst sich daher mit der Frage, wie eine Simulationsanwendung und die entsprechenden Spezifikationen von Szenarien aufgebaut sein müssen, um die Verifizierung und Validierung maritimer automatisierter Fahrsysteme und -funktionen entwicklungsbegleitend zu ermöglichen.

Um dieses Ziel zu erreichen, wurde auf ein zweiteiliges ingenieurwissenschaftliches Vorgehen gesetzt. Zunächst wurde das technologische und methodische Aufgabenumfeld ausführlich aufgearbeitet. Dabei lag der Fokus auf der ganzheitlichen und harmonisierenden Betrachtung der Themenkomplexe *Simulation*, *szenariobasiertes Testen*, *automatisierte maritime Fahrzeuge* sowie *Verifizierung und Validierung*. Auf Basis von abgeleiteten übergreifenden Anforderungen an eine potenzielle Lösung wurde der aktuelle Stand der Technik anhand eines systematischen Literaturreviews erhoben und bezüglich bestehender Probleme szenariobasierter simulativer Testprozesse analysiert.

Eine der zentralen Herausforderungen besteht in der Auflösung der bisher überwiegend anzutreffenden orthogonalen Beziehung der Modellierungsräume von Szenariomodell und Simulationsmodell, mit welcher eine starke Kopplung und eine geringe Flexibilität gängiger Modelle und Anwendungen einhergehen. Im Zentrum des erarbeiteten mehrteiligen Lösungsansatzes steht die automatisierte, unmittelbare und vollständige Erzeugung des Ausgangszustands eines Simulationslaufs aus einer Szenarioinstanz. Zu diesem Zweck wurden ein auf szenariobasierte Simulationsausführung ausgerichtetes Metamodell sowie eine auf diesem aufbauende, mehrschichtige Architektur zur flexiblen, komponentenbasierten Modellierung von Verkehrsszenarien erarbeitet. Die im IEEE-Standard 1516-2010 definierte High-Level-Architecture bildet die strukturelle Basis der ausführenden Simulationsanwendung und wurde durch stringente Komplexitätskapselung, automatisierte Orchestrierung sowie flexibel konfigurierbare Kommunikation mit dem simulationsexternen Testobjekt erweitert.

Durch eine umfangreiche abschließende Untersuchung einer prototypischen Umsetzung der erarbeiteten Konzepte, dem Framework *Distributed Component-Based Traffic Simulation (DisCo-BaTS)*, konnte u. a. im Rahmen einer Fallstudie die Eignung der Lösungskonzepte für den zukünftigen praktikablen Einsatz von szenariobasierten simulativen Testreihen im Rahmen der Verifizierung und Validierung maritimer Assistenzsysteme untersucht und bestätigt werden. Durch die gebotene Flexibilität, Erweiterbarkeit und Wiederverwendbarkeit stellt das Framework eine angemessene Grundlage für zukünftige einheitliche Umsetzungen virtueller Testfelder vielfältiger Ausprägungen dar und leistet somit einen wertvollen Beitrag auf dem Weg zum breiten Einsatz hochautomatisierter Wasserfahrzeuge.

ABSTRACT

Public acceptance is vital for the widespread introduction of highly automated ships. The requisite public confidence in the pertinent technologies can only be fully achieved through the assurance of functionality and safety. The overall increase in the complexity of the systems, caused by the advent of electronic and software-based systems and functions, in conjunction with the unique characteristics of contemporary ship development processes, has given rise to a series of novel challenges in the domain of safety verification. One promising approach is scenario-based testing. Computer simulations are regarded as the most pragmatic approach to conducting corresponding test series. However, existing simulation applications are reaching their limits, and there is a lack of scenario-based simulation applications that can be flexibly integrated as testing tools into a wide range of development and testing processes.

This work, therefore, addresses the question of how a simulation application and the corresponding specifications of scenarios must be designed to enable the verification and validation of maritime automated driving systems and functions during development.

To achieve this objective, a two-part scientific engineering approach was adopted. Initially, the technological and methodological environment of the task to fulfill was examined in detail. The emphasis was placed on a holistic and harmonizing consideration of topics like *simulation*, *scenario-based testing*, *automated maritime vehicles,* and *verification and validation*. A systematic literature review was conducted to survey the current state of the art, and the results were analyzed regarding existing problems of scenario-based simulation test processes.

A central challenge is to resolve the orthogonal relationship between the modeling spaces of scenario models and simulation models. This has been prevalent to date and is associated with strong coupling and low flexibility in common models and applications. The multi-part solution approach developed is aligned on the automated, immediate, and complete generation of the initial state of a simulation run from a scenario instance. For this purpose, a metamodel directly targeting scenario-based simulative testing and a multi-layered architecture based on this metamodel were developed for flexible, component-based modeling and execution of traffic scenarios. The high-level architecture defined in IEEE Standard 1516-2010 is used as the basis of the executing simulation application and has been extended by stringent complexity encapsulation, automated orchestration, and flexibly configurable communication with the test object outside the simulation.

A comprehensive evaluative investigation was conducted using the prototypical implementation of the developed concepts, the *Distributed Component-Based Traffic Simulation (DisCo-BaTS)* framework. This investigation confirmed the suitability of the solution concepts for the practical use of scenario-based simulative test series in the context of verification and validation of maritime assistance systems. This was achieved through a case study, among other methods. The framework's flexibility, expandability, and reusability provide an appropriate basis for future uniform implementations of virtual test fields of various types, thus making a valuable contribution to reaching the goal of widespread use of highly automated ships.

ANMERKUNG

Im Rahmen dieser Arbeit wird aus Gründen der einfacheren Lesbarkeit im Kontext eines komplexen Themas bei personenbezogenen Formulierungen auf die Geschlechterunterscheidung verzichtet. Es wird alternativ auf das aktuell noch zweckmäßige generische Maskulinum zurückgegriffen. Dies soll keinesfalls als Diskriminierung von Geschlechtern oder Personengruppen aufgefasst werden, sondern dient lediglich der sprachlichen Vereinfachung. Ich bin mir bewusst, dass die aus dieser Entscheidung resultierenden Formulierungen für einige Personen herabsetzend und verletzend wirken können, und bitte darum, mir dies im Rahmen der vorliegenden Verschriftlichungen meiner wissenschaftlichen Bemühungen der letzten Jahre zu verzeihen.

INHALT

Kurzfassung III
Abstract V
Anmerkung VII
Abkürzungen und Akronyme XI

TEIL I – EINLEITUNG 1

1 Sicherheit und Akzeptanz ... 3
2 Problembeschreibung ... 6
3 Forschungsfragen ... 10
4 Betrachtungsumfang ... 13
5 Forschungsdesign ... 18
5.1 Explorative und konstruktive Phase ... 19
5.2 Aufbau und Methodenauswahl ... 20

TEIL II – V&V SOFTWAREBASIERTER MARITIMER FAHRSYSTEME 25

6 Entwicklung moderner Seeschiffe ... 26
6.1 Automatisierung und Autonomie ... 26
6.2 Schiffsführungssysteme ... 33
6.3 Systembegriff ... 36
6.4 Vorgehensmodelle ... 44
7 Beweis von Funktionalität und Sicherheit ... 46
7.1 Verifizierung und Validierung ... 47
7.2 Szenariobasiertes Testen ... 51
7.3 Computergestützte Testdurchführung ... 61
7.4 Modellierung von Szenarien ... 81
8 Simulationsanwendungen anderer Verkehrszweige ... 88
9 Anforderungen an maritime Simulationsanwendungen ... 90
10 Aktueller Einsatz maritimer Simulationen zu Testzwecken ... 93
10.1 Erhebungsmethode ... 93
10.2 Durchführung ... 97
10.3 Ergebnisse ... 98
11 Handlungsbedarf ... 100

TEIL III – KOMPONENTENBASIERTE MODELLIERUNG VON VERKEHRSSZENARIEN 103

12 Gestaltungsorientierte Zielsetzung ... 104
13 Lösungsbezogene Anforderungen ... 107
13.1 Nutzergruppen szenariobasierter Simulationsläufe ... 108
13.2 Anforderungen an die softwaretechnische Lösung ... 110
13.3 Zusammenfassung ... 115
14 Konstruktiver Lösungsentwurf ... 115
14.1 Lösungskonzepte ... 116
14.2 Verwandte Lösungsansätze ... 133
14.3 Inhaltliche Ausgestaltung ... 136

15 Experimentelle Umsetzung 165
15.1 Simulationsanwendung 167
15.2 Modelle 170
15.3 Orchestrierung 174
16 Überprüfung und Bewertung 176
16.1 Untersuchung der Anforderungserfüllung des Prototyps 176
16.2 Untersuchung der Eignung des Prototyps 181
16.3 Untersuchung des Mehrwerts des Prototyps 193
16.4 Zusammenfassende Untersuchung der Zielerreichung 194

TEIL IV – ZUSAMMENFASSUNG 197

17 Einordnung des Forschungsbeitrags 198
18 Reflexion der entwickelten Lösung 199
19 Anknüpfender Handlungs- und Forschungsbedarf 200

Literatur 205
Abbildungen 225
Tabellen 231
Listings 233
Veröffentlichungen 235
Anhang 237

ABKÜRZUNGEN UND AKRONYME

ABM	Agent Based Modelling
AIS	Automatic Identification System
API	Application Programming Interface
ATON	Aid to Navigation
BDI	Beliefs, Desires, and Intentions
BNF	Backus Naur Form
CAN	Controller Area Network
COLREG	International Regulations for Preventing Collisions at Sea
CPA	Closest Point of Approach
CPS	Cyber-Physical System
DDM	Data Distribution Management
DES	Discrete-Event Simulation
DisCo-BaTS	Distributed Component-Based Traffic Simulation
DSEEP	Distributed Simulation Engineering and Execution Process
DSL	Domain-Specific Language
eMIR	e-Maritime Integrated Reference Platform
FEDEP	Federation Development and Execution Process
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
FOM	Federation Object Model
FTP	File Transfer Protocol
GGA	Global Positioning System Fix Data
GNC	Guidance, Navigation, and Control
GPS	Global Positioning System
HiL	Hardware-in-the-Loop
HLA	High Level Architecture
JAR	Java Archive
JAXB	Jakarta XML Binding
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KI	künstliche Intelligenz
LA	Lösungsorientierte Anforderung
LAN	Local Area Network
LRC	Local RTI Component
maRTIx	Model-Aware RTI-Exchange
MAS	Multiagentensimulation
MASS	Maritime Automated Surface Ships
MiL	Model-in-the-Loop
MOM	Management Object Model
MTCAS	Maritime Traffic Alert and Collision Avoidance System
NMEA	National Marine Electronics Association
OMT	Object Model Template
OOP	Object Oriented Programming

RTI	Runtime Infrastructure
SDL	Scenario Description Language
SiL	Software-in-the-Loop
SM	Systemmodell
SOM	Simulation Object Model
SoS	System-of-Systems
SOTIF	Safety of the Intended Functionality
SSH	Secure Shell
SUT	System Under Test
TAR	Time Advance Requests
TCP	Transmission Control Protocol
TSM	Teilsystemmodell
TZ	Teilziel
UDP	User Datagram Protocol
UML	Unified Modeling Language
V&V	Verifizierung und Validierung
ViL	Vehicle-in-the-Loop
VT	Verkehrsteilnehmer
WAN	Wide Area Network
XiL	X-in-the-Loop
XML	Extensible Markup Language
Z	Ziel

"Remember that all models are wrong;
the practical question is how wrong do they have to be to not be useful."

George E. P. Box and Draper R. Norman (1987)
„Empirical Model Building and Response Surfaces"

»Ce qui est simple est toujours faux. Ce qui ne l'est pas est inutilisable.«

„Was simpel ist, ist immer falsch. Was komplex ist, ist unbrauchbar."
(frei übersetzt)

A. Paul T. J. Valéry (1942)
„Mauvaises Pensées et autres"

I Einleitung

„no trust, no use“
[Schaefer et al., 2016]

Die Entwicklung automatisierter Fahrzeuge hat in den vergangenen Jahren erhebliche Fortschritte gemacht [Costa et al., 2025; Munim & Haralambides, 2022]. In der öffentlichen Wahrnehmung nimmt der Automobilsektor dabei eine führende Position unter den Verkehrsdomänen ein. Begründet ist diese partiell durch eine gewisse historische und kulturelle Leitrolle, die diesem zugeschrieben werden kann [Kröger, 2016]. Aber auch Transportsysteme anderer Domänen erfahren ein zunehmendes Interesse in Bezug auf Automatisierung und angestrebte Autonomie.

Mit Blick auf die Wirtschaft und die stetig weiter voranschreitende Globalisierung dieser fällt den maritimen Transportsystemen eine ganz besondere Rolle zu. Aufgrund der dominierenden Rolle des Seewegs im weltweiten Warentransport und des stetig wachsenden Handelsvolumens auf selbigem sind maritime Transportsysteme von höchster Wichtigkeit für moderne Lieferketten und somit die globale Wirtschaft [Özkanlisoy & Akkartal, 2022; UNCTAD, 2020]. Auch für lokale Volkswirtschaften, vorrangig in der Nähe von großen Überseehäfen, ist der Warentransport auf dem Seeweg zu einer tragenden Säule geworden. So beziffert [Bräuninger et al., 2021] die Anzahl der direkt oder indirekt vom Hamburger Hafen abhängigen Beschäftigungen im Jahr 2019 mit 606.700 unter Einbeziehung der gesamten hafenabhängigen Wertschöpfungsketten. Betrachtet man die deutschlandweite Bedeutung aller See- und Binnenhäfen, wird die Bedeutung dieser noch einmal deutlicher: Im Jahr 2017 existierten laut [Lemper et al., 2019] insgesamt 8.882.600 direkt und indirekt hafenabhängige Beschäftigungen, was annähernd 10 % der Einwohneranzahl Deutschlands entspricht. Der gesamte von deutschen Häfen abhängige Umsatz belief sich dabei 2016 auf 172,8 Mrd. Euro [Bräuninger et al., 2021]. Diese Zahlen sind so beträchtlich, dass die Häfen und die Seeschifffahrt als systemrelevant für die deutsche Wirtschaft anzusehen sind [Lemper et al., 2019; BMWi, 2017]. Zusammenfassend lässt sich daher schlussfolgern, dass die Wirtschaft sowohl global als auch regional auf zuverlässige und effiziente maritime Transportsysteme angewiesen ist.

Als zusätzlichen starken Innovationstreiber der jüngeren Zeit lässt sich das gestiegene Bewusstsein für die Wichtigkeit des Schutzes der natürlichen Umwelt und des Klimas vor schädlichen menschlichen Einflüssen identifizieren [Fisahn, 2018]. Aus diesem gestiegenen Umweltbewusstsein resultierten nicht nur erhebliche öffentliche, wirtschaftliche und wissenschaftliche Anstrengungen [Wright, 2020], sondern auch Bestrebungen und Beschlüsse zum Zwecke des Umwelt- und Klimaschutzes auf staatlicher Ebene wie u. a. das Klimaabkommen von Paris [UN, 2015], die es nötig machen, die Schifffahrt, als größten Träger des weltweiten Warentransports, zukünftig auch in Hinblick auf Themen wie den Ressourceneinsatz, die Lärmemission sowie die Schadstoffemissionen effizienter zu gestalten.

Obwohl der maritime Sektor häufig als eher konservativ in Bezug auf Innovationen beschrieben wird [Koukaki & Tei, 2020; Wright, 2020; Acciaro et al., 2018], ist es daher nicht verwunderlich, dass auch das wissenschaftliche Interesse an maritimen Themen wie autonomer Schifffahrt, Navigationsalgorithmen und (maritimer) Kollisionsvermeidung im Laufe der letzten Jahre stark angestiegen ist

[Hatledal, 2021; Porres et al., 2020b]. Das deutsche Bundesministerium für Wirtschaft und Energie (BMWi) betont ebenfalls die Wichtigkeit von Digitalisierung maritimer Verkehrssysteme sowie explizit auch die Entwicklung von Assistenzsystemen für die automatisierte Schifffahrt zur Förderung von Effizienz, Sicherheit und Nachhaltigkeit in seiner *Maritimen Agenda 2025* [BMWi, 2017]. Darauf aufbauend wurde in Zusammenarbeit mit Vertretern der maritimen Branche die maritime Forschungsstrategie 2025 ausgearbeitet, welche unter anderem folgende strategische Ziele definiert [BMWi, 2018]:

- Erhöhung der Sicherheit in der Schifffahrt
- Steigerung der Transport- und Schiffsbetriebseffizienz
- Nachhaltige Transportkonzepte und Geschäftsmodelle

Auf dem Weg zur Erreichung dieser Ziele nehmen die maritime Digitalisierung, neuartige hochautomatisierte Assistenzsysteme sowie die angestrebte vollständige Autonomie von Schiffen eine Schlüsselrolle ein: Assistenzsysteme können Schiffsbewegungen überwachen und werden zunehmend dafür eingesetzt, kritische Situationen zu erkennen und zukünftig ohne menschliches Eingreifen zu vermeiden. Auf Basis digitaler, global verfügbarer Informationen könnten Geschwindigkeiten, Kurse und Routen so angepasst werden, dass die Betriebskosten und Emissionen auf ein technisch mögliches Minimum gesenkt werden [Khabir et al., 2025; Wright, 2020; Zanella, 2020]. Autonome Fahrzeuge als oberstes Ziel der Forschung und der Industrie könnten dann wiederum eine solche optimale Route selbst identifizieren, umsetzen und bei Bedarf in Echtzeit anpassen. Neben einem solchen automatisierten globalen Transport von Waren und Gütern sind zusätzlich kleinere maritime Transportsysteme denkbar, wie Flotten kleiner autonomer Wasserfahrzeuge, welche eingesetzt werden könnten, um auch den Warentransport weiter ins Inland hinein auf effiziente Weise zu ermöglichen.

Eine der größten Herausforderungen bei der Entwicklung von hochautomatisierten und autonomen Transportsystemen ist der Nachweis ihrer fehlerfreien Funktion. Gerade für die erfolgreiche Einführung zukünftiger Schiffe mit einer Autonomiestufe von drei oder höher in der Kategorisierung der *International Maritime Organization (IMO)* [IMO, 2018], ist der Nachweis ihrer korrekten Funktionsfähigkeit und Sicherheit unabdingbar. Die vergleichbaren Autonomiegrade im Automobilbereich sind die SAE-Level 3 und höher [SAE, J3016:JAN2014], für welche bereits neue Qualitätsstandards sowie Testwerkzeuge und -methoden vorgeschlagen wurden, u. a. im Rahmen der deutschen Forschungsprojekte PEGASUS und SET Level [DLR, 2022, 2019]. Der Nachweis der Sicherheit steht auch deshalb im Fokus, da dieser einer der Schlüssel zur öffentlichen Akzeptanz hochautomatisierter und autonomer Fahrzeugsysteme ist, wie eine Studie der deutschen und amerikanischen Printmedien gezeigt hat [Lenz & Fraedrich, 2015]. Der nachvollziehbare, glaubhafte Nachweis von Sicherheit stärkt letztlich nachvollziehbarerweise das Vertrauen der Allgemeinheit in derartige Systeme. Ist dieses Vertrauen nicht vorhanden, wird die Nutzung entsprechender Systeme häufig gemieden [Kim & Oviedo-Trespalacios, 2025], die flächendeckende Einführung stärker automatisierter Systeme verlangsamt [Damsara & Barros, 2025; Hoppe et al., 2024] und mehr Zeit und Geld in die aktive Kontrolle der Systeme investiert, als nötig wäre [Mansfield et al., 2022]. Die US-Bundesbehörde *National Highway Traffic Safety Administration* formuliert diese Tatsache noch etwas drastischer: Das Ausbleiben öffentlicher Akzeptanz derartiger Systeme hat das Potenzial, die flächendeckende Einführung dieser stark zu behindern oder gar gänzlich zu verhindern [NHTSA, 2017].

1 SICHERHEIT UND AKZEPTANZ

Die Sicherheit von Fahrzeugen ist eines der wichtigsten Kriterien bei der Entwicklung dieser. Neben der Unabdingbarkeit für eine öffentliche Zulassung, der Akzeptanz der potenziellen Nutzer und der intrinsischen Motivation, ein gutes Produkt zu fertigen, haben Entwickler, Hersteller und Vertreiber solcher Systeme auch existenzielle Gründe, ihre Systeme möglichst sicher zu gestalten: Sollte ihr Produkt aufgrund von Design-, Entwicklungs- oder Produktionsfehlern Schäden an Menschen oder Material verursachen, zieht dies meist finanzielle Aufwände und einen Imageverlust (vgl. u. a. [Penmetsa et al., 2021]) nach sich. Wie schwerwiegend die finanziellen Folgen in so einem Fall sein können, konnte in der jüngsten Zeit beobachtet werden, nachdem zwei Passagierflugzeuge des Typs Boeing 737-8 Max kurz nach dessen Markteinführung abgestürzt waren. Der Hersteller Boeing schätzte Anfang 2020, dass die Gesamtkosten der Abstürze und des anschließenden Flugverbots der betroffenen Flugzeugtypen mehr als 18,6 Milliarden Dollar betragen werden [Khashe & Levy, 2020]. In dieser Summe sind noch keine Strafzahlungen aus Rechtsverfahren oder langfristige Verluste durch Imageschäden enthalten. Mitverantwortlich für die Abstürze war ein Fehlverhalten des softwarebasierten Assistenzsystems *Maneuvering Characteristics Augmentation System (MCAS)*. Das Ziel, eine hohe Softwarequalität mithilfe von Tests sicherzustellen, hat demnach nicht nur einen idealistischen Hintergrund, sondern ist auch getrieben durch mögliche ökonomische Auswirkungen [Majchrzak, 2012].

Wichtig für die weitere Betrachtung der Sicherheit von neuartigen Fahrzeugsystemen ist dabei zunächst die genaue Analyse des Begriffes *Sicherheit*. Dieser spaltet sich in mehrere Unterbereiche auf, welche sich vor allem als Leser von deutscher Literatur auf den ersten Blick nur schwer voneinander unterscheiden lassen. So wird u. a. für die englischen Begriffe *Safety* und *Security* in der deutschen Sprache beide Male der kontextabhängige Begriff *Sicherheit* als übliche Übersetzung herangezogen. Auch unterscheiden sich die international verbreiteten Interpretationen von *Safety* teilweise voneinander. Nachfolgend werden an dieser Stelle basierend auf größtenteils englischsprachiger Literatur, Normen und Standards einige der wichtigsten Teilbereiche von *Sicherheit* für den weiteren Verlauf dieser Arbeit definiert. Aufgrund des aktuell hohen Forschungs- und Veröffentlichungsvolumens im Bereich der Straßenfahrzeuge wird sich hauptsächlich an den dort vorzufindenden Definitionen bedient. Die Beziehungen der verschiedenen betrachteten Sicherheitsbegriffe untereinander sind in Abbildung 1 ergänzend grafisch dargestellt. Verbindungen zur internationalen Norm für die Entwicklung von computerbasierten Anwendungen auf Schiffen *ISO 17894* [ISO, 17894:2005] werden zusätzlich erwähnt,

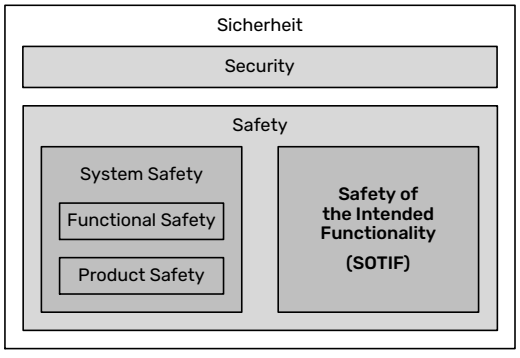


Abbildung 1: Begriffe mit Bezug zum Konzept der *Sicherheit* in ihren Menge-Teil-Beziehungen dargestellt. Das für diese Arbeit relevanteste Konzept der *Safety of the Intended Functionality (SOTIF)* ist durch Fettdruck hervorgehoben.

um aufzuzeigen, dass die Definition auch auf den maritimen Bereich übertragen werden kann.

Sicherheit beschreibt als Oberbegriff einen Zustand, in dem Gefahren und Bedingungen, die zu physischen, psychischen oder materiellen Schäden führen können, soweit kontrolliert werden oder gar abwesend sind, dass die Gesundheit und das Wohlbefinden des Einzelnen, der Gemeinschaft und der Umwelt erhalten bleiben. [Formela et al., 2019]

Security beschreibt den Grad, zu dem ein Produkt oder System Informationen und Daten schützt, so dass Personen, andere Produkte oder Systeme den Grad des Datenzugriffs erhalten, der ihrer Art und Berechtigungsstufe entspricht [Ross, 2016]. Kurz gefasst kann man hier sagen, dass Security bedeutet, dass die Systeme sowie Strukturen und Daten innerhalb dieser vor unberechtigtem Zugriff geschützt sind. Eine Risikoquelle kann etwa eine Fehlfunktion sein, die durch eine unberechtigte Manipulation des Systems entsteht, bei der Daten und Strukturen innerhalb der Software verändert werden. Diese Veränderungen könnten, vom Angreifer bewusst oder unbewusst, zu einem sicherheitskritischen Verhalten des Fahrzeugs führen [Glas et al., 2015]. Analoges Prinzip für marine programmierbare elektronische Systeme (engl.: programmable electronic systems, PES) aus ISO 17894: „P7 Unauthorized access to the PES shall be prevented."

Safety adressiert den Schutz der Benutzer und der Umwelt vor dem Fehlverhalten des Systems, d. h., von der Benutzung des Systems darf keine größere Gefährdung ausgehen, als es nach dem Stand von Wissenschaft und Technik vermeidbar ist [Klauda et al., 2012]. Dies deckt sich wohl am ehesten mit dem, was im deutschen Sprachgebrauch allgemein unter *Sicherheit* verstanden wird. Je nach Kontext kann *Safety* aber nach wie vor unterschiedliche Implikationen für die Entwicklung haben: Bei einem Unfall sollen die Insassen des Verkehrsmittels möglichst gut geschützt sein, automatisierte Systeme sollen keine vermeidbaren Fehler machen, Benutzer sollen sich nicht durch materialistische Verarbeitungsfehler bei der Nutzung verletzen können u. v. m. Daher ist es auch hier nötig, wieder kontextabhängig zwischen Unterbereichen zu unterscheiden. Analoges Prinzip für marine PES aus ISO 17894: „P1 – The PES [Programmable Electronic System] shall be free from unacceptable risk of harm to persons or the environment."

↳ **Functional Safety:** Betrachtet man die Domäne der Straßenfahrzeuge, so kommt man an dem Begriff der *Functional Safety* (dt.: funktionale Sicherheit) und der ISO-Norm 26262 [ISO, 26262-1:2018] nicht vorbei. Innerhalb der Norm wird die funktionale Sicherheit wie folgt definiert: die Abwesenheit von inakzeptablem Risiko aufgrund von Gefahren, die durch fehlerhaftes Verhalten von elektrisch-elektronischen (E/E) Systemen verursacht werden. Der Begriff umfasst also nur die Aufrechterhaltung der Sicherheit, z. B. durch implementierte Rückfallebenen (engl.: fallback), im Falle eines bereits eingetretenen Fehlers. Ein nicht softwarebasiertes klassisches Beispiel ist hier die Servolenkung von Personenkraftwagen: Fällt die hydraulische Unterstützung aus, so kann das Fahrzeug trotzdem noch ausreichend gefahrlos gelenkt werden, um an einem sicheren Ort zu halten. ISO 26262 schließt dabei funktionale Leistungen explizit dadurch aus, dass Funktionen, die bereits zu einer Gefährdung führen, ohne dass ein Fehler aufgetreten ist, generell nicht betrachtet werden [Miller, 2020]. Analoges Prinzip für marine PES aus ISO 17894: „P2 – In the event of failure, the PES shall remain in or revert to the least hazardous condition."

↳ **Safety of the intended Functionality:** Der funktionalen Sicherheit steht die sogenannte *Safety of the intended Functionality* (SOTIF, dt.: Sicherheit der vorgesehenen Funktionalität) ergänzend gegenüber.

Diese deckt den Bereich der Sicherheit ab, den die funktionale Sicherheit explizit ausklammert. Die ISO-Norm 21448 definiert SOTIF wie folgt: „The absence of unreasonable risk due to hazards resulting from functional insufficiencies of the intended functionality or by reasonably foreseeable misuse by persons is referred to as the Safety Of The Intended Functionality (SOTIF)." [ISO/PAS, 21448:2019] Hier wird also explizit die Funktion der Systeme betrachtet, ohne dass zuvor Software- oder Hardwarefehler eingetreten sind. Zusammenfassend kann man sich im Rahmen der SOTIF die Frage stellen, ob das entwickelte System in der vorgesehenen Umgebung und unter vorgesehenen Umständen so funktioniert und agiert, wie es bei dem Entwurf und der Entwicklung geplant war. Entsprechendes Prinzip für marine PES aus ISO 17894: „P1 b) – The acceptable risk associated with the PES should be based on its intended use, covering all defined operating conditions, and reasonably foreseeable misuse."

↳ **Product Safety:** Die funktionale Sicherheit und die SOTIF beziehen sich beide ausschließlich auf funktionale Abläufe von Fahrzeugsystemen. Die Produktsicherheit setzt an derselben Stelle an wie die SOTIF, ergänzt dabei aber die Sicherheit in der Abwesenheit von Fehlern, grob zusammengefasst, um die physische Sicherheit, wie das Fehlen von scharfen Kanten oder die Sicherheit der Mensch-Maschine-Schnittstelle. [Miller, 2020]

↳ **System Safety:** Die Sicherheit des Systems beschreibt abschließend die Gesamtsicherheit für den Benutzer und die Umwelt während der Nutzung des Systems und umfasst die funktionale Sicherheit sowie die Produktsicherheit. [Miller, 2020]

Betrachtet man die Sicherheit aus der Perspektive der klassischen Produktentwicklung, so kann diese als ein essenzieller Teil der zu Beginn der Entwicklungsphase zu identifizierenden Anforderungen angesehen werden. Um die jeweilige Sicherheit zu beweisen, benötigt es somit umfangreiche Tests, wie es auch für die restlichen aufgestellten Anforderungen der Fall ist. Die gängige Praxis im Bereich der Entwicklung neuartiger Fahrzeugsysteme ist es, derartige Tests im Rahmen von umfangreicheren Methoden zur Verifizierung und Validierung durchzuführen. Verifizierung und Validierung sind jene Methoden, mit deren Hilfe möglichst zuverlässige Aussagen darüber zu treffen versucht wird, ob ein Produkt oder ein System den definierten Spezifikationen entspricht und den vorgesehenen Zweck erfüllt. Dabei ist die Verifizierung individuell betrachtet ein Qualitätskontrollverfahren, mit welchem bewertet wird, ob ein Produkt, eine Dienstleistung oder ein System den Vorschriften und Anforderungen entspricht, die zu Beginn einer Entwicklungsphase identifiziert und festgelegt wurden. Bei der Validierung hingegen handelt es sich um einen Qualitätssicherungsprozess, bei dem Nachweise erbracht werden, die ein hohes Maß an Sicherheit darüber bieten sollen, dass ein Produkt oder ein System für den beabsichtigten Gebrauch geeignet ist und die reellen Bedarfe der Akteure deckt [Maropoulos & Ceglarek, 2010]. Die Begriffe *Verifizierung* und *Validierung* sind zwar getrennt voneinander definiert, doch kann der größtmögliche Nutzen nur erbracht werden, wenn sie synergetisch eingesetzt werden [Wallace & Fujii, 1989]. Daher wird *Verifizierung und Validierung* (engl.: Verification and Validation, V&V) heutzutage im Kontext von Software-, System- oder Produktentwicklung in der Regel als eine integrierte Einheit verstanden, definiert und angewendet.

Die starke Anreicherung von Fahrzeugen mit E/E und softwarebasierten Systemen und Systemkomponenten führt dazu, dass vor allem die funktionale Sicherheit und die SOTIF in den vergangenen Jahren stark in den Fokus gerückt sind. Besonders in Bezug auf fortschrittliche Assistenzsysteme mit

hohem Automatisierungs- und Autonomiegrad. Konkret ist dabei insbesondere für die erfolgreiche Einführung zukünftiger Schiffe mit einem Autonomiegrad von drei oder mehr, wie von der IMO kategorisiert [IMO, 2021a], und zukünftiger Straßenfahrzeuge mit einer Fahrautomatisierung der Stufe 3 oder höher, wie von der SAE definiert [SAE, J3016:APR2021], der Nachweis der korrekten und sicheren Funktionsweise unabdingbar, da der Mensch als kontrollierende und im Zweifelsfall eingreifende Instanz weniger bis gar nicht mehr vorgesehen ist. Die Relevanz der veränderten Sicherheitsbetrachtung kann dabei aufgrund der anzunehmenden zukünftigen Zunahme eben solcher Systeme [Weber et al., 2020] im öffentlichen Raum, nicht von der Hand gewiesen werden.

Standards und Normen aus dem maritimen Sektor, wie IMO 1580 [IMO, MSC.1/Circ.1580], orientieren sich bisher stark am Konzept der funktionalen Sicherheit. In der Automobilindustrie werden Automatisierungssysteme inzwischen sowohl für die funktionale Sicherheit nach der ISO-Norm 26262 als auch für die SOTIF nach der ISO-Norm 21448 zertifiziert, und es ist wahrscheinlich, dass eine ähnliche Entwicklung zeitnah auch in der maritimen Industrie stattfinden muss – spätestens jedoch für die Einführung vollautonomer Schiffe (engl. Maritime Autonomous Surface Ships, MASS) [Torben et al., 2022]. Dies ist unter anderem darin begründet, dass zusammen mit steigendem Automatisierungsgrad die Anzahl softwarebasierter Systeme auf einem Schiff ebenfalls zunimmt. Zusätzlich zu dieser Steigerung der Komplexität des Gesamtsystems werden auch die einzelnen Teilsysteme mit zunehmender Übertragung von Aufgaben und Verantwortungen auf diese komplexer. Durch die steigende Komplexität der Systeme selbst sowie der Abhängigkeiten zwischen diesen besteht die Gefahr ungewollter und unvorhersehbarer Wechselwirkungen zwischen den Teilkomponenten [Klauda et al., 2012].

> *„[…] it can be argued that software systems grow faster in size and complexity than methods to handle complexity are invented.“*
>
> [Majchrzak, 2012]

Gleichzeitig werden die höheren Automatisierungsgrade zu einem großen Teil durch den Einsatz von Technologien und Verfahren aus den Bereichen der *künstlichen Intelligenz* (KI) und des *maschinellen Lernens* (ML) erreicht, deren Verhalten gar nicht oder zumindest nicht vollständig deterministisch beschrieben werden kann [Dreossi et al., 2019; Mallozzi et al., 2019]. Derartige neue Systeme und Systemzusammenschlüsse erfordern somit gänzlich neue Wege der Sicherheitsbewertung und der technischen Methoden, einschließlich der Verifikations- und Validierungsmethoden und -technologien [Wright, 2020; Hahn et al., 2015; Thomke & Fujimoto, 2000].

> *„We have not found any […] description on how to test AI or ML systems. This is an indication that this field is far from ready for validation of […] safety critical systems.“*
>
> [Myklebust & Stålhane, 2021]

2 PROBLEMBESCHREIBUNG

Wie in den vorigen Abschnitten bereits angeführt, hält die in der Automobil- und Luftfahrtbranche schon länger präsente und schnell fortschreitende Digitalisierung und Automatisierung von Fahrzeugen auch im modernen Schiffbau langsam, aber sicher, Einzug. Diese Entwicklung führt zwangsläufig zu einer Zunahme der Komplexität des Gesamtsystems *Schiff* [Hatledal, 2021]. Insbesondere auf der Brücke ist die Digitalisierung bereits angekommen: Hier sind mittlerweile eine Vielzahl an unter-

schiedlichen elektronischen Komponenten, z. B. zur Unterstützung bei der Navigation und Entscheidungsfindung der Besatzung, zu finden, welche zusätzlich häufig mittels Netzwerkkommunikation Daten untereinander austauschen und zu einem sogenannten *Integrierten Brückensystem* (engl. Integrated Bridge System, IBS) zusammengefasst werden [Liu et al., 2014].

Werften, die über die letzten Jahrzehnte vielfach bereits einen Wandel von Produzenten hin zu Systemintegratoren durchlebt haben [Held, 2010; Gronau et al., 2003], bedienen sich dabei, um dem angestiegenen Aufwand Herr zu werden, in der Regel einer großen Menge an Zulieferern für den Entwurf, die Entwicklung, die Produktion und die Auslieferung von diesen softwarebasierten Komponenten. Somit werden diese zunächst als eigenständige Systeme umgesetzt und anschließend mit den Systemen anderer Zulieferer in das Gesamtsystem integriert [Gomes et al., 2019]. Um dem durch die fortschreitende Automatisierung und den Einsatz von Echtzeit-Warnsystemen gestiegenen Bedarf an Echtzeitdaten gerecht zu werden, werden moderne Schiffe zusätzlich mit zahlreichen Sensoren und Sensordaten verarbeitenden Systemen angereichert [Durlik et al., 2025]. Vor allem aufgrund der so entstandenen Unabhängigkeit und Eigenständigkeit dieser parallel entwickelten Komponenten, sowohl während des Betriebs als auch während der Entwicklung, sollte ein Schiff nicht mehr als ein monolithisches Einzelsystem betrachtet werden. Stattdessen wird im Rahmen dieser Arbeit die Auffassung vertreten, dass ein modernes Schiff, als die Gesamtmenge der zusammenarbeitenden Teilsysteme, ein sogenanntes *System-of-Systems* (SoS) [Keating et al., 2003] darstellt.

Die rasante Zunahme an softwarebasierten Systemen und der gleichzeitig, wie in allen Verkehrsdomänen, angestrebte hohe Automationsgrad, bis hin zur vollständigen Autonomie als übergreifendes Ziel, stellen die produzierenden Unternehmen und die Werften als Systemintegratoren vor bisher nicht dagewesene Herausforderungen. Der ehemalige Secretary-General und CEO des *Baltic and International Maritime Council* (*BIMCO*[1]), Angus Frew, sprach 2017 in einem Interview ebenfalls davon, dass dem Management softwarebasierter Systeme in der maritimen Industrie bisher nicht ausreichend Beachtung geschenkt wurde:

> *„The industry has been living in a world of hardware. But software has been integrated into most […] vessels, and the systems and procedures to manage the software has not kept up with technical developments, and it creates problems."*
>
> [Jorgensen, 2017]

Die Einführung nicht-expliziter Ansätze, wie dem *maschinellen Lernen* (engl.: Machine Learning) aus dem Fachbereich der *künstlichen Intelligenz* (KI), und der daraus hervorgehenden impliziten Modelle hat dafür gesorgt, dass sich ein Großteil klassischer statischer formaler Test- und Sicherheitsprüfmethoden, wie Modellprüfung und maschinengestütztem Beweisen, als untauglich in Bezug auf den Beweis der Sicherheit von automatisierten Fahrzeugsystemen herausgestellt hat [Boudardara et al., 2023; Karthaus et al., 2021; Singh & Saini, 2021]. Formale Methoden stützen sich zumeist stark auf das vollständige Wissen über das zu prüfende Modell. Ein maschinell gelerntes Modell hingegen ist von außen nicht nachvollziehbar und ähnelt eher einer undurchsichtigen Blackbox [Eschenbach, 2021], und das Verhalten der Systeme, die solche Modelle einsetzen, kann daher oft nur zur Laufzeit überprüft wer-

[1] BIMCO ist laut eigener Aussage der größte Zusammenschluss unabhängiger Schiffseigner, Reedereien, Schiffsmakler, Schifffahrtsagenturen und zahlreicher anderen Schifffahrtsbeteiligter. Damit geht eine gewisse Vorreiterrolle einher, anhand welcher globale Einflussnahme auf zukünftige Entwicklungen möglich sind.

den[2]. Hinzu kommt, dass die bereits erwähnte gestiegene Komplexität und Vernetzung solcher Systeme dazu führen, dass die Systemgrenzen zunehmend verschwimmen und die Funktionsprüfung eines einzelnen isolierten Assistenzsystems nicht mehr ausreicht [Mallozzi et al., 2019; Brinkmann et al., 2017].

Eine Möglichkeit, diesen neuen Anforderungen gerecht zu werden, ist die Durchführung von Testläufen des integrierten Gesamtsystems inklusive aller Bestandteile. In der Automobilbranche werden solche Testfahrten klassischerweise am Ende des Entwicklungs- und Integrationsprozesses durchgeführt, und auch im Schiffsbau wären derartige Testfahrten denkbar, da die elektronischen Komponenten durchaus noch ausgebessert oder ausgetauscht werden können, nachdem das Schiff zu Wasser gelassen wurde. Allerdings ist es unmöglich, alle denkbaren Situationen, denen ein automatisiertes oder teilautonomes Fahrzeug in Zukunft ausgesetzt sein könnte, manuell zu testen – was aufgrund des nichtdeterministischen Verhaltens nötig wäre.

Die Abdeckung aller Permutationen von Umgebungsvariablen durch reale Testfahrten zu garantieren, erfordert einen enormen Einsatz von Zeit und Ressourcen, was den Test- und infolgedessen den Entwicklungsaufwand zukünftiger Systeme unhaltbar kostenintensiv machen würde [Wachenfeld & Winner, 2015b]. Um dieses Problem anzugehen, werden derzeit mögliche neue Methoden für die Freigabe von hochautomatisierten und autonomen softwarebasierten Transportsystemen erforscht und entwickelt. Ein sehr populärer Ansatz, um Testfahrten in der realen Welt zu ersetzen, ist die Verwendung von Verkehrssimulationen [Klikovits et al., 2024; Lou et al., 2022; Porres et al., 2020a; ISO/PAS, 21448:2019; Takács et al., 2018; Barceló, 2010a]. Die Durchführung von Testfahrten in einer simulierten virtuellen Umgebung bringt eine Vielzahl an Vorteilen mit sich: die Fähigkeit, bereits während früher Phasen der (Software-)Entwicklung eingesetzt werden zu können, schneller als in Echtzeit ausgeführt werden zu können, und Menschen, Umwelt und Material keinem Risiko auszusetzen u. v. m. [Bossel, 2004]

Aber auch unter dem Einsatz von Simulationen als Substitution für die reale Welt ist die virtuell vom zu testenden System zu fahrende Strecke weiterhin so lang, dass eine vollständige Testabdeckung im Rahmen einer Neuentwicklung nicht realistisch erreichbar ist [Lou et al., 2022]. Dies trifft umso mehr zu, je automatisierter das Fahrzeug ist, und gipfelt in vollständig autonomen Systemen. Dabei ist die Strecke, welche in alltäglichem realistischem Verkehr nötig ist, zusätzlich noch einmal umso länger, je besser und sicherer ein autonomes Fahrzeug ist [Kalra & Paddock, 2016]. Um konkrete Zahlen für autonome Straßenfahrzeuge zu ermitteln, haben Kalra & Paddock basierend auf aktuellen Unfallstatistiken Hochrechnungen mit verschiedenen Einflussfaktoren durchgeführt und sind zu dem Ergebnis gekommen, dass, um mit 95 % Konfidenz und einer Teststärke von 80 % zu beweisen, dass ein autonomes Straßenfahrzeug in Bezug auf tödliche Unfälle 20 % sicherer ist als ein menschlicher Fahrer, insgesamt 11 Milliarden Testkilometer von diesem zurückgelegt werden müssten. Das entspricht bei einer Testflotte von 100 Fahrzeugen, die ohne Unterbrechung mit einer Durchschnittsgeschwindigkeit von ca. 40 km/h am realen Straßenverkehr teilnehmen, einer Testdauer von 500 Jahren. Auch wenn das Anwendungsgebiet stark eingeschränkt wird, bleibt die zu fahrende Strecke beachtlich. So

[2] Es existieren zahlreiche Ansätze, welche versuchen die Abläufe innerhalb von maschinell gelernten Modellen, verständlich, erklärbar und somit vertrauenswürdig zu machen (u.a. [Zednik, 2021]). Diese werden im Allgemeinen im Bereich der *Explainable Artificial Intelligence (XAI)* zusammengefasst, welcher aktuell einen sehr hohes Forschungsinteresse erfährt. Alle bisher vorgestellten Ansätze stellen aber u.a. laut [Mittelstadt et al., 2019] lediglich limitierte approximative lokale Modelle zur Erklärung der ML-Modelle zur Verfügung. 8

wurde für einen autonomen Autobahnpiloten errechnet, dass dieser im Rahmen von Testfahrten 6,62 Milliarden Autobahnkilometer zurücklegen müsste [Wachenfeld & Winner, 2016, 2015a, 2015b]. Da die Länge der zu fahrenden Strecke stark von der durchschnittlichen Wahrscheinlichkeit des Auftretens von kritischen Situationen, beinahe Unfällen und Unfällen abhängt, kann angenommen werden, dass sich die notwendigen Strecken und zeitlichen Aufwände zur Absicherung maritimer Fahrsysteme und Fahrzeuge noch einmal drastisch erhöhen: Schiffe, vor allem solche, die hauptsächlich in offenen, weitläufigen Gewässern unterwegs sind, wie die für unsere Wirtschaft so wichtigen Containerschiffe, begegnen deutlich seltener anderen Schiffen und Hindernissen im Vergleich zu einem durchschnittlichen Straßenfahrzeug.

Softwaresimulationen können meist schneller als Echtzeit ausgeführt werden, wodurch dieses Problem auf den ersten Blick keines mehr zu sein scheint. Allerdings bringt die aktuell verfügbare Technologie Einschränkungen mit sich: Ab einem gewissen Punkt begrenzen maximale Schreib- und Lesegeschwindigkeiten von Speichermedien oder die maximale Rechenleistung der Prozessoren die Geschwindigkeit, mit der maximal simuliert werden kann. Selbst unter der häufig realitätsfernen Annahme, dass man die Simulation mit einem Faktor von 1000 schneller durchführen kann als Echtzeit, beträgt die Dauer für den oben beschriebenen Beweis der Sicherheit statt 500 Jahren noch immer 0,5 Jahre. Für die effiziente Entwicklung von solchen modernen Systemen ist es nicht realistisch, einzelne Entwicklungsschritte oder Updates jeweils ein halbes Jahr lang zu testen – der Zeitaufwand für die Entwicklung und Markteinführung neuer Systeme würde in einem Umfang ansteigen, welcher diese für kein profitorientiertes Unternehmen mehr rentabel umsetzbar machen würde.

Um effizient aussagekräftige Ergebnisse aus simulativen Tests zu erhalten, ist also zusätzlich ein systematischer Ansatz erforderlich [Lou et al., 2022; Lamm & Hahn, 2018]. Ein populärer Ansatz, der diese Anforderung erfüllt, ist *szenariobasiertes Testen*, wie es unter anderem in [Porres et al., 2020a; Akkermann & Hjollo, 2019; DLR, 2019; Brinkmann et al., 2017] vorgeschlagen wird. Ein szenariobasiertes Vorgehen zeichnet sich dadurch aus, vorab Verkehrssituationen zu identifizieren, die für die Aussagekraft der Simulationsergebnisse in Bezug auf die Sicherheit eines bestimmten zu prüfenden Fahrsystems (System Under Test, SUT) relevant sind. Die so entstandene Menge an relevanten Situationen kann anschließend modelliert, simuliert und bewertet werden. Aufgrund der Popularität und des Erfolgs im Automobilbereich kann davon ausgegangen werden, dass ein szenariobasierter Ansatz auch für die simulative V&V von Assistenzsystemen im maritimen Bereich erfolgreich eingesetzt werden kann. Ein allgemeiner Überblick auf hohem Abstraktionsniveau über die Hauptaktivitäten eines solchen simulations- und szenariobasierten Ansatzes und deren Zusammenhänge ist in Abbildung 2 dargestellt.

Moderne Schiffe werden selten monolithisch entworfen und konstruiert, vielmehr werden sie durch die Integration einer Vielzahl heterogener Teilsysteme unterschiedlichster Hersteller als System-of-Systems erzeugt. Softwarebasierte Systeme können und sollten dabei bereits entwicklungsbegleitend getestet werden, um hohe Kosten zu vermeiden [Witte, 2016]. Auch die Norm ISO 17894 weist darauf hin, dass Verifizierungs- und Validierungsmaßnahmen während des gesamten Lebenszyklus von *programmierbaren elektronischen Systemen* (PES) durchgeführt werden sollten, um das angestrebte Sicherheitsniveau in jeder Entwicklungsphase zu realisieren [ISO, 17894:2005]. Durch diese beiden Tatsachen ergibt sich somit zusätzlich die Notwendigkeit, softwarebasierte elektronische Teilsysteme eines

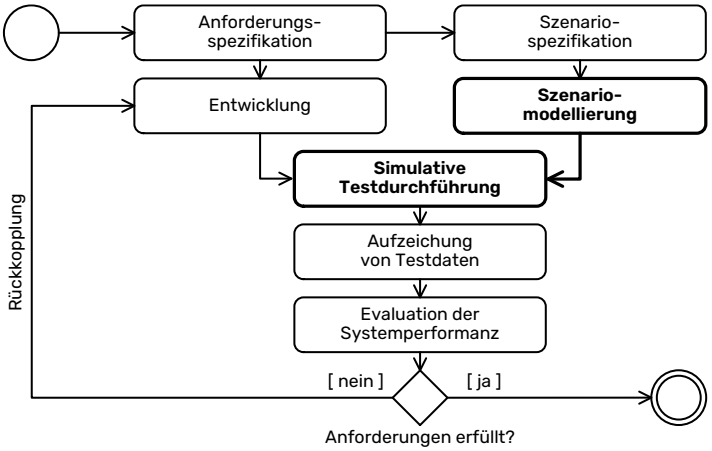


Abbildung 2: Einordnung der in dieser Arbeit betrachteten Szenariomodellierung in den szenariobasierten Entwicklungsprozess von automatisierten Fahrsystemen (grob angelehnt an den in [Hanke, 2020] vorgestellten Prozess). Die Teilprozesse, die Hauptgegenstand dieser Arbeit sind, sind fett umrandet.

Fahrzeugs unabhängig voneinander im Rahmen der individuellen Entwicklungsphasen simulativ testen zu können. Dafür muss die Möglichkeit geboten werden, Simulationsbestandteile einfach, die Anforderungen der aktuellen Entwicklungsphase erfüllend, kombinieren und parametrieren zu können und Komponenten eines simulierten Fahrzeugs durch ein oder mehrere reale Systeme, externe Simulationen, externe Software oder externe Modelle ersetzen zu können.

3 FORSCHUNGSFRAGEN

Für die erfolgreiche Entwicklung und öffentliche Einführung von maritimen automatisierten und autonomen Fahrsystemen bedarf es Simulationsanwendungen für maritime Verkehrsszenarien, welche es ermöglichen, die im vorigen Abschnitt 2 angeführten Herausforderungen an die Verifikation und Validierung dieser Systeme effizient zu bewältigen. Dabei scheint die Systematisierung des generellen Vorgehens durch den Einsatz eines szenariobasierten Ansatzes der beste Weg zu sein, um die Problematik des ansteigenden Testaufwands in den Griff zu bekommen.

Um im Rahmen von Verifizierung und Validierung (V&V) szenariobasierte Simulationsläufe sinnvoll einsetzen zu können, ist es notwendig, die Ausgangszustände und Systemanregungen im Voraus genau zu definieren, um die Basis dafür zu schaffen, bestimmte Zustandskombinationen gezielt simulieren, beobachten und auswerten zu können. Der Ausgangszustand für die Durchführung einer Simulation enthält verschiedene Bestandteile, wie die Verkehrsteilnehmer, die Umwelt, in der diese sich bewegen, statische Hindernisse, Verhaltensregeln und Beziehungen zwischen diesen.

Unterschiedliche zu testende softwarebasierte Systeme in unterschiedlichen Phasen im Softwareentwicklungsprozess erfordern unterschiedliche Grade an Spezifität, Komplexität und Granularität der Simulation (vgl. [Klikovits et al., 2024]). Wird etwa ein fiktives Assistenzsystem zur Vermeidung von Kollisionen betrachtet, so kann man sich leicht vorstellen, dass für erste Testläufe der generellen Entscheidungsfindung in einem frühen Entwicklungsstadium möglicherweise eine Repräsentation eines Verkehrsteilnehmers als Punkt in Form einer einzelnen Koordinate ausreicht. Für Testläufe im Rahmen der Fertigstellung des Assistenzsystems sollte die simulierte Umgebung dann aber möglichst realitätsnah abgebildet sein – Schiffe also u. a. eine dreidimensionale Form haben, welche über Sensoren erfasst werden kann. Ein anderes zu testendes System könnte nun zusätzlich noch gänzlich andere In-

formationen wie den Tiefgang eines Schiffes voraussetzen. Um die abgebildeten Situationen und die Reaktionen simulierter Verkehrsteilnehmer auf die erwähnten Anregungen jeweils auf das zu testende softwarebasierte System und die aktuelle Phase im Softwareentwicklungsprozess anpassen zu können, muss die Möglichkeit einer Anpassbarkeit der Simulationsanwendung und der Szenarien an die Anforderungen des konkreten zu testenden Systems gegeben sein. Um den individuellen Aufwand dabei trotzdem gering zu halten, muss eine hohe Wiederverwendbarkeit ermöglicht werden und Szenarien sowie ihre Bestandteile müssen persistiert, übertragen und verwendet werden können.

Aus den in Abschnitt 2 beschriebenen und hier exemplarisch konkretisierten Herausforderungen an die Verifizierung und Validierung moderner softwarebasierter maritimer Fahrsysteme ergibt sich übergreifend die folgende Forschungsfrage, deren Beantwortung sich diese Arbeit widmet:

Leitfrage

Wie können Szenarien unterschiedlicher Komplexität und Granularität im Bereich der maritimen Verkehrssimulationen als Basis für die jeweiligen Simulationsläufe modelliert, übertragen und simuliert werden, um die Verifizierung und Validierung neuer softwarebasierter Systeme entwicklungsbegleitend zu ermöglichen?

Aus den durch die Leitfrage adressierten Inhalten und der vorausgegangenen ersten Beschreibung der Herausforderungen ergeben sich drei Teilfragen, die gezielt unterschiedliche Aspekte der Leitfrage vertiefen. Diese drei nachfolgend beschriebenen Teilfragen decken dabei die drei großen identifizierten Anforderungsbereiche ab. Die Erfüllung der jeweiligen Anforderungen liefert konkrete Zwischenergebnisse, welche gemeinsam wiederum zur Beantwortung der Leitfrage beitragen. Die gewählte Unterteilung soll so dazu beitragen, ein zielgerichtetes Vorgehen zu unterstützen. Dabei umfasst jede individuelle Teilfrage die Identifizierung relevanter Anforderungen, die Frage nach einem zugrunde liegenden abstrakten Konzept und eine mögliche technische Gestaltung.

Teilfrage 1

Wie müssen Simulationsanwendungen und Szenariomodelle aufgebaut sein, damit sichergestellt werden kann, dass diese möglichst weitreichend zu Verifizierungs- und Validierungszwecken von maritimen Fahrsystemen eingesetzt werden können?

Simulationsanwendungen für maritime Fahrsysteme existieren in unterschiedlichen Ausprägungen: Monolithische Simulationsanwendungen können von einfachen Prototypen, welche im Rahmen wissenschaftlicher Bemühungen entstanden und häufig sehr spezialisiert auf die Beantwortung einer spezifischen Frage ausgelegt sind, bis zu schwergewichtigen Werkzeugen zur Analyse von Verkehrsflüssen [Reiher & Hahn, 2021a] reichen. Verteilte Simulationen koppeln heterogene, alleinstehende, möglicherweise stark spezialisierte, Simulationsanwendungen auf eine Weise, die es ermöglicht, dass diese zusammen an einem Simulationslauf teilnehmen können. Eine besondere Rolle nehmen zusätzlich noch solche Simulatoren ein, welche spezifische Hardware umfassen und hauptsächlich für Trainings- und Schulungszwecke von Schiffspersonal eingesetzt werden [Reiher & Hahn, 2021a]. Solche *Schiffsführungssimulatoren* sind vergleichbar mit Flugsimulatoren, in welchen Piloten simulative Trainingsflüge als Teil ihrer Ausbildung absolvieren. Im Rahmen dieser Arbeit stellt sich nun die Frage, welche zentralen Eigenschaften eine Simulationsanwendung haben muss, um für das hier angestrebte Ziel der weitreichenden Unterstützung des Verifizierungs- und Validierungsprozesses moderner maritimer

Fahrsysteme entwicklungsbegleitend einsetzbar zu sein. Ebenso stellt sich die Frage, ob verfügbare Technologien und Ansätze die Eigenschaften bereits aufweisen, und sollte das nicht der Fall sein, welche Teilziele sich aus den aufgedeckten Lücken für diese Arbeit ergeben.

TEILFRAGE 2

Wie kann die Erstellung von Szenarien für Verkehrssimulationen so unterstützt werden, dass dieser Prozess möglichst wenig domänenunabhängiges Wissen im Bereich der zugrunde liegenden Szenariomodellierung und der genutzten Simulationsanwendung voraussetzt?

Für den effizienten Einsatz während des gesamten Entwicklungszyklus von Assistenzsystemen sollte es den Entwicklungsteams möglich sein, zum Zweck der frühen Aufdeckung von Fehlern oder unerwartetem Verhalten, Szenarien zu erstellen, die an den aktuellen Entwicklungsstand angepasst sind. Der nötige Aufwand, um Testszenarien mit bestehenden Ansätzen zu erstellen, ist einer der Hauptkritikpunkte von Anwendern von Simulation im Umfeld von automatisierten Fahrsystemen [Lou et al., 2022]. Um die breite Akzeptanz über den gesamten Entwicklungszyklus zu fördern und den zusätzlichen Arbeitsaufwand für Szenarien unterschiedlicher Abstraktionsgrade gering zu halten, sollte für diese Tätigkeit kein spezifisches technisches Wissen über die Simulationsanwendung selbst benötigt werden. Es stellt sich daher für diese Arbeit die Frage, wie die Erstellung von Szenarien so gestaltet werden kann, dass möglichst keine Anpassungen an der Simulationsanwendung selbst durchgeführt werden müssen, obwohl sich die Anforderungen an die Inhalte, die Komplexität und die Granularität der Testszenarien für unterschiedliche Entwicklungsphasen und unterschiedliche zu testende Assistenzsysteme stark unterscheiden können.

TEILFRAGE 3

Wie kann ermöglicht werden, dass heterogene zu testende Systeme am Simulationslauf teilnehmen können, ohne dass Änderungen oder Anpassungen an der Simulationsanwendung oder am zu testenden System selbst vorgenommen werden müssen?

Um eine Simulationsanwendung für das entwicklungsbegleitende Testen eines neuen Assistenzsystems und die Verifikation und Validierung dessen einsetzen zu können, ist es unabdingbar, dass das zu testende System an den Simulationsläufen teilnehmen kann. Das inkludiert primär die Möglichkeit des zu testenden Systems, auf Inputs der Simulation reagieren zu können und auf Inhalte der Simulation aktiv einwirken zu können. In unterschiedlichen Entwicklungsphasen existiert ein solches zu testendes System in unterschiedlich abstrakten Zuständen: von einem ersten einfachen Modell über ausführbaren Programmcode unterschiedlicher Fertigstellungsgrade, bis zu fertiggestellten Systemen aus Hardware und Software, welche als Blackbox[3] betrachtet werden. Im Sinne von Teilfrage 2 muss die Integration des zu testenden Systems dabei auf der Ebene eines Szenarios möglich sein. Für die vorliegende Arbeit stellt sich daher die Frage, wie zu testende Systeme so in die Szenarien integriert werden können, dass sie bei der Ausführung dieser aktiv am Simulationslauf teilnehmen können, ohne dass jeweils Änderungen an der eigentlichen Simulationsanwendung vorgenommen werden müssen.

[3] In der Systemtheorie wird ein System, von dem nur das äußere Verhalten beobachtet werden kann, als Blackbox bezeichnet. Die innere Struktur ist nicht bekannt oder wird nicht betrachtet. Tests einer solchen Blackbox werden auch als funktionale Tests bezeichnet. Hierbei werden die Testfälle allein auf der Grundlage der Systemspezifikation entworfen und die Untersuchungen beschränken sich auf die Messung der Input-Output-Beziehungen unter bestimmten Ausführungsbedingungen. [Nidhra, 2012]

4 BETRACHTUNGSUMFANG

Die vorliegende Arbeit konzentriert sich auf technische Konzepte für die Simulation von maritimem Verkehr und die Modellierung von Szenarien als Ausgangspunkt dieser. Das übergreifende Themenfeld der szenariobasierten simulationsgestützten V&V ist vielfältig und komplex. Diese Arbeit erhebt nicht den Anspruch, dass das Ergebnis allein ausreichend ist, damit einer erfolgreichen Einführung von automatisierten Wasserfahrzeugen nichts mehr im Weg steht. Stattdessen leistet die Beantwortung angrenzender Fragestellungen einen ebenso wichtigen und unverzichtbaren Beitrag zu diesem übergreifenden Ziel. Um die präsentierten Teillösungen vorab klar abzugrenzen, aber auch um die Wichtigkeit der aktiven Arbeit an der Vielzahl verwandter Themen zu betonen, folgt eine Auflistung und Beschreibung der Themengebiete, die in dieser Arbeit nicht betrachtet werden. Dabei wird jeweils begründet, warum diese Entscheidung getroffen wurde und wo die Grenze der Betrachtung gezogen wurde.

3D-Visualisierung: Bei Verwendung des Begriffs *Simulation* hat man häufig unmittelbar eine dreidimensionale grafische Repräsentation der simulierten Welt auf großen Bildschirmen vor dem inneren Auge, da vor allem Trainings- und Ausbildungssimulatoren der Öffentlichkeit bekannt sind. Diese benötigen aufgrund der Mensch-Simulation-Interaktion eine möglichst detaillierte und realitätsnahe dreidimensionale grafische Darstellung, damit die Trainingssituation einer realen Situation möglichst ähnlich ist. Da die Einbindung von grafischen Darstellungen die Komplexität der Simulation und der nötigen Modelle beträchtlich erhöht [Chwif et al., 2000], wird eine 2D- oder 3D-Visualisierung vorerst im Rahmen dieser Arbeit nicht betrachtet. Dadurch wird der wissenschaftliche Wert dieser Arbeit nicht gemindert, da die Fragestellung im Kern eine architekturelle ist und der gezeigte Ansatz in Bezug auf eine computergrafische Visualisierung erweiterbar ist.

Vollständige Sensorsimulationen: Eine umfängliche Modellierung und Simulation von Sensoren ist notwendig, um moderne automatisierte Assistenzsysteme und Teilsysteme autonomer Fahrzeuge zu testen, da diese über ebensolche Sensoren ihre Umwelt wahrnehmen (vgl. u. a. [Durlik et al., 2025]). Ist ein simulativ zu testendes System auf Daten solcher Sensoren als Input angewiesen, so müssen diese Daten ebenfalls in der Simulation zur Verfügung stehen und an das zu testende System geleitet werden, damit dieses sich während des Simulationslaufs so verhält, wie es dies in der Realität täte. Aufgrund der Vielzahl von Simulationen und der Komplexität der virtuellen Abbildung von Sensoren wird der umgesetzte Prototyp keine Sensormodelle enthalten, die bereit für einen produktiven Einsatz sind. Auch wird das Thema keinen zentralen Aspekt dieser Arbeit darstellen und nicht in der Tiefe betrachtet werden, wie es zum Beispiel in [Schweigert, 2017] der Fall ist. Das Lösungskonzept lässt allerdings eine entsprechende Erweiterung zu, sodass die Nichtbeachtung den Wert der Ergebnisse nicht schmälert. Zentrale Entscheidungen während der Erarbeitung der Konzepte sowie der prototypischen Implementierung wurden stets in Hinblick auf eine solche Erweiterbarkeit getroffen, auch wenn die Modellierung und Implementierung von Sensoren nicht explizit adressiert wird.

Identifizierung von Szenarien: Eine der zentralen Fragestellung in Bezug auf die Verifikation und Validierung (V&V) von automatisierten und autonomen Fahrzeugen [Lou et al., 2022; Pedersen et al., 2020], welche sich grob mit folgenden Unterfragen beschreiben lässt: Wie kann bestimmt werden, welche Szenarien für aussagekräftige Testläufe geeignet sind? Wie bestimmt man die Aussagekraft eines Szenarios? Wie können diese automatisiert erzeugt werden? Diese Fragen eröffnen ein ganz eigenes

und ebenfalls sehr komplexes Forschungsgebiet, welches aber keine direkten Berührungspunkte mit dem Inhalt dieser Arbeit hat. Lou et al. [2022] nutzen für diese Aufgabe den Begriff *Corner Case Generation* und bezeichnen es auf Basis einer nicht-systematisch durchgeführten Literaturanalyse als das aktuell meistbeachtete Forschungsfeld im Gebiet des Testens von automatisierten Fahrsystemen (engl.: Automated Driving System, ADS). Aufgrund der klaren inhaltlichen Grenze, des Umfangs dieser Fragestellung und der großen Aufmerksamkeit, die ihr bereits zukommt, wird dieses Thema nicht betrachtet. Ziel dieser Arbeit ist es, im Anschluss an die Identifizierung anzuknüpfen und eine gute technische Grundlage für die persistente Speicherung der Szenarien sowie für iterative Suchverfahren zu bieten. Aufgrund der Relevanz des Themas sei dennoch auf einige Arbeiten verwiesen, in denen unterschiedliche Ansätze verfolgt werden. Die Liste der genannten Arbeiten ist dabei keineswegs vollständig und soll nur einige Beispiele aufzeigen.

Die älteste Gruppe von Ansätzen ist dabei wohl die der datengetriebenen. In diesen werden die zu testenden Szenarien aus bereits vorhandenen Datenaufzeichnungen extrahiert. Als Datenbasis dient dabei etwa eine Menge aufgezeichneter Fahrdaten, die während realer Testfahrten oder im normalen Alltag aufgezeichnet wurden [Lamm, 2023; Elspas et al., 2020; Geiger et al., 2012]. Aber auch offizielle Unfallberichte können als Datenbasis dienen [Gambi et al., 2019]. Die Verwendung von Unfallberichten geht dabei meist mit der Hoffnung einher, die Menge des „Rauschens“ in den nötigen Daten zu reduzieren, da in diesen lediglich kritische und daher für Tests relevante Situationen enthalten sind. Weil Unfallberichte in den allermeisten Fällen nicht alle nötigen Daten enthalten, um ein vollständiges Testszenario abzuleiten, werden trotzdem häufig die erstgenannten Ansätze genutzt, welche aber zusätzlich vorab reduziert oder eingeschränkt werden. Dies kann zum Beispiel auf Basis von Expertenwissen geschehen [Wurst et al., 2022]. Zur Reduzierung der auszuwertenden Datenmenge wurden aber auch bereits vielfach Ergebnisse von Arbeiten veröffentlicht, welche verschiedenste Kombinationen von heterogenen Datenquellen kombinieren. Dabei wird auch die Vollständigkeit der abgeleiteten Szenarien wahrscheinlicher. [Pütz et al., 2017]

Um den manuellen Aufwand weiter zu reduzieren, fand in den jüngeren Jahren eine Vielzahl von Methoden aus dem Bereich der *künstlichen Intelligenz* Anwendung. Darunter Ansätze wie das *maschinelle Lernen* [Ding et al., 2021], Lösungsalgorithmen für *Constraint-Satisfaction-Probleme* [Karimi & Duggirala, 2022] und *neuronale Netze* [Ben Abdessalem et al., 2016]. Aus dem Teilbereich des maschinellen Lernens sind vertiefend auch das *Clustering* historischer Daten und anschließendes *Sampling* zur Erzeugung konkreter Szenarien [Wüllner et al., 2019] erwähnenswert. Auch existieren Arbeiten, die wiederum versuchen, mehrere Ansätze der genannten Bereiche optimierend zu kombinieren [Blache et al., 2023].

Bewertung von Szenarien: Eine thematische Teilmenge des vorigen Punktes stellt die Bewertung vorhandener Szenarien dar. Um kritische oder aussagekräftige Szenarien, in Bezug auf ein konkretes zu testendes System, zu erzeugen, muss es möglich sein, deren Kritikalität oder Aussagekraft zu quantifizieren. Dies ist nötig, um eine Vergleichbarkeit von konkreten Szenarien zu ermöglichen, um sowohl automatisierte Erzeugungsansätze zu steuern, als auch händisch geeignete Szenarien auszuwählen. Beispielhaft seien an dieser Stelle zwei generelle Ansätze genannt: die Identifizierung und Quantifizierung von konkreten Gefahren und Risiken innerhalb des Verlaufes von Szenarien, wie sie in [Salem et al., 2024] und [Westhofen et al., 2023] bearbeitet wurden, sowie die generelle Bewertung der Qualität

eines Szenarios als Ganzes, wie sie unter anderem in [Kolb et al., 2022] betrachtet wird. Da der vorgestellte Ansatz nicht zum Ziel hat, Szenarien zu erzeugen, sondern vielmehr als Tool für die automatisierte Erzeugung genutzt werden kann (vgl. voriger Absatz), befindet sich auch die mögliche Betrachtung einer automatisierten Bewertung von Szenarien außerhalb des Umfangs der vorliegenden Arbeit.

Vollständige Modelle: Das Ziel dieser Arbeit ist es, ein Simulationswerkzeug zu entwerfen, welches für Tests im Rahmen der Verifikation und Validierung von automatisierten Systemen und Teilsystemen moderner maritimer Fahrzeuge eingesetzt werden kann (vgl. Abschnitt 12). Eine der übergreifenden Herausforderungen hierbei ist die hohe Diversität der Anforderungen unterschiedlicher zu testender Systeme in unterschiedlichen Entwicklungsphasen (vgl. Abschnitt 7.1.1). Das erarbeitete Konzept sieht daher eine strikte Trennung der Modelle, der Szenario-Beschreibung und der Simulationsumgebung vor (vgl. Abschnitte 14.1.4 und 14.1.8). Angesichts dessen ist es nicht sinnvoll, ein umfangreiches Modell zu liefern, wie es etwa bei der offenen Simulationsumgebung für autonomes Fahren CARLA[4] der Fall ist[5]. Da auch die Physiksimulation und steuerungsbezogene Algorithmen, wie konkrete Fahralgorithmen und Routenplanungsalgorithmen, in die anwendungsfallspezifischen Modelle ausgelagert sind, werden auch diese keine besondere Aufmerksamkeit erfahren, die über den für die Evaluation nötigen Prototypenstatus hinausgeht.

Parameteroptimierung: Für einen Teil der Testfälle ist es nötig, dass die Simulation möglichst realitätsgetreu abläuft, da ein zu testendes System, zumindest in den späten Entwicklungsschritten, Inputwerte erwartet, die der Realität entsprechen. Damit dies möglich ist, ist es in der Regel nötig, das Simulationsmodell zu kalibrieren. [Liu et al., 2024a] Der Prozess der Kalibrierung von Simulationsparametern ist die methodische Bestimmung der optimalen Kombination von Parametern für das verwendete Modell mit dem Ziel, die Diskrepanz zwischen den Simulationsausgangsdaten und den realen Beobachtungsdaten zu verringern [Burger et al., 2013]. Da in der vorliegenden Arbeit lediglich ein prototypisches Modell zum Zweck des Beweises der Funktionalität umgesetzt wurde, bestand auch kein Bedarf an einer vertiefenden Betrachtung der Optimierung der Parameter dieses Modells. Die Optimierung der Parameter ist modellabhängig und obliegt somit dem Entwickler des jeweiligen Modells (vgl. Abschnitt 13.1).

Validierung von Modellen: Die Validierung der zur Simulationsdurchführung genutzten Modelle überschneidet sich mit dem vorigen Punkt der Parameteroptimierung. Gültigkeit, Glaubwürdigkeit, Nützlichkeit und Praktikabilität sind sowohl für einfache als auch für zusammengesetzte Modelle zentrale Anforderungen. Da diese Arbeit nicht den Anspruch hat, unmittelbar produktiv einsetzbare Modelle zu liefern, liegt die Validierung dieser außerhalb des Betrachtungsumfangs. Auch werden keine Methoden geliefert, mit denen erstellte Modelle direkt auf Korrektheit überprüft werden können. Dies zu tun, liegt in der Hand der Person, die die Modelle mithilfe des entworfenen Ansatzes erstellt (vgl. Abschnitt 13.1). Als Hilfestellung in Form eines guten ersten Einblicks in das Thema sei an dieser Stelle aber auf [Denil, 2023] verwiesen. Dort wird ein erster Überblick über verschiedene Ansätze zur Prüfung der Validität von Modellen, die Gefahren für die Validität von Simulationsmodellen und einige Tools zur Unterstützung bei der Validierung von Simulationsmodellen gegeben.

[4] `https://carla.org/` [letzter Zugriff: 17.02.26]

[5] `https://carla.readthedocs.io/en/latest/catalogue/` [letzter Zugriff: 17.02.26]

Digitale Zwillinge: Oftmals werden die Themen *Simulation* und *digitaler Zwilling* vermischt betrachtet, obwohl die Begriffe klar disjunkte Definitionen besitzen. Da das Ziel dieser Arbeit ist, Simulationsläufe für entwicklungsbegleitende Tests zu ermöglichen, wird keine Garantie darauf gegeben, verlässliche digitale Zwillinge, wie sie unter anderem in [Ali et al., 2024; Dirnfeld et al., 2024; Giering & Dyck, 2021; Singh et al., 2021] definiert und betrachtet werden, mit dem hier präsentierten Ergebnis erstellen zu können. Begründung findet diese Tatsache unter anderem in der fehlenden Echtzeitfähigkeit und der theoretisch möglichen, aber nicht explizit konzipierten und nicht erprobten, engen bidirektionalen Kopplung aller Komponenten eines realen Fahrzeugs mit jeweils einem digitalen Abbild dieser Komponente [Giering & Dyck, 2021]. Zudem ist nicht gesichert, dass die Auflösung der Modelle, der Daten und der Übertragungsrate den Ansprüchen eines digitalen Zwillings genügt.

Auswertung von Simulationsläufen: Simulationsläufe, die zu Testzwecken durchgeführt wurden, müssen anschließend ausgewertet werden, damit diese ihren vorgesehenen Zweck erfüllen können. Wie auch für die Erstellung der Modelle gilt für die eigentliche Auswertung, dass das vorgestellte Simulationswerkzeug die dafür nötigen Mittel, in Form von Schnittstellen, Konfigurationen etc., zur Verfügung stellt, aber keine konkreten Methoden bietet, da diese ebenfalls stark anwendungsfallabhängig sind. So steht unter anderem eine Möglichkeit zur Verfügung, die zu loggenden Daten zu definieren, welche dann entsprechend während des Simulationslaufs persistent gespeichert oder zur Laufzeit flüchtig ausgegeben werden können. Die weiterführende Analyse dieser Daten, inklusive aller notwendigen Verarbeitungsschritte wie der Aggregation, ist kein Bestandteil dieser Arbeit.

Weiterführende Sicherheitsuntersuchungen: Für eine erfolgreiche Einführung automatisierter und autonomer Fahrzeuge ist es unabdingbar, auch weitere sicherheitsrelevante Aspekte neben der *Sicherheit der beabsichtigten Funktionalität* (vgl. Abschnitt 1) zu überprüfen. So steigt die Bedeutung von Sicherheit im Sinne des englischsprachigen Begriffs *Security* mindestens linear mit der Zunahme der Menge softwarebasierter Systeme in Fahrzeugen: Angriffe, wie die unbefugte Kontrolle von Fahrzeugfunktionen aus der Entfernung und das unbefugte Auslesen von Sensordaten, müssen zur Sicherheit der Insassen und der Umgebung unbedingt verhindert werden. Auch die klassisch präsente *funktionale Sicherheit* hat trotz der Anreicherung mit Software nicht an Bedeutung verloren. Beide Bereiche werden bereits durch andere Testverfahren abgedeckt, sodass in dieser Arbeit der Fokus ausschließlich auf der oben genannten *Sicherheit der beabsichtigten Funktionalität* liegt. Für eine ausführliche Diskussion aktueller und zukünftiger Herausforderungen im Bereich der *Security* automatisierter und autonomer Schiffe sei auf [Wright, 2020] verwiesen.

Unzugängliche Lösungen: Bei der Betrachtung von verwandten Arbeiten in Abschnitt 10 werden keine Lösungen betrachtet, deren Quellcode oder die zugrundeliegenden Konzepte nicht zugänglich sind. Die Einsehbarkeit der zugrundeliegenden Strukturen ist unabdingbar für das Verständnis der Abläufe und das zentrale wissenschaftliche Qualitätskriterium der Überprüfbarkeit [Balzert et al., 2022; Heesen, 2021]. Dadurch ist auch ein großer Teil der aktuell am Markt verfügbaren kommerziellen und teilkommerziellen Lösungen von der Betrachtung ausgeschlossen. Eigenständige Veröffentlichungen zu solchen Lösungen unterlagen in der Regel keinem Peer-Review-Verfahren, wodurch die wissenschaftlich relevante Objektivität der Inhalte nicht nachgewiesen werden kann. Diese Veröffentlichungen können daher ebenfalls nicht vollständig in Betracht gezogen werden. Um Lösungen dieser Art

nicht in Gänze zu ignorieren und die softwaretechnischen Leistungen angemessen zu würdigen, werden nachfolgend einige Beispiele in einer unvollständigen Liste genannt:

↳ **dSpace ASM-Produktfamilie[6]:** Die *Automotive Simulation Models* Produktfamilie bietet verschiedene Anwendungen zur Modellierung und Simulation von Straßenverkehr und einzelnen Fahrzeugen. Dabei wird überwiegend auf grafische Desktop-Anwendungen gesetzt, um die Bedienbarkeit einfach zu halten. Es werden unter anderem Produkte angeboten für Simulationssteuerung, Simulationsparametrierung, Erstellung von Fahr- und Verkehrsszenarien und die Simulationsdurchführung selbst.

↳ **IPG Automotive CarMaker[7]:** Die unter dem Namen *CarMaker* als Simulationslösung angebotene Zusammenstellung separater Anwendungen und Tools wurde gezielt für entwicklungsbegleitende Tests von Pkw und leichten Nutzfahrzeugen entwickelt. Dabei werden alle gängigen X-in-the-Loop Varianten unterstützt, die Erstellung von Szenarien unterstützt, eine grafische 3D-Umgebung sowie diverse Schnittstellen geboten, und die Testauswertung durch integrierte Analysetools unterstützt. In [Graubohm et al., 2023] wird CarMaker eingesetzt, um im Rahmen der Evaluation ein entworfenes dynamisches Geschwindigkeitsmodell zu testen.

↳ **Chiastek CosiMate[8]:** *Co-Simulation Interface (CosiMate)* ist ein Softwaretool zur nahtlosen Kopplung unterschiedlicher Simulationsumgebungen und -anwendungen. Es ermöglicht den Austausch und die Synchronisierung von Daten und eignet sich somit für komplexe Systeme, wie sie in der Automobil- und Luftfahrtindustrie üblich sind. *CosiMate* unterstützt die Effizienz des Systementwurfs durch die Möglichkeit der Integration einer Reihe von ansonsten sehr spezialisierten, dedizierten Simulationswerkzeugen.

↳ **Siemens PreScan[9]:** Um die Entwicklung von Fahrerassistenzsystemen zu beschleunigen, bietet die physikbasierte Simulationsplattform *PreScan* eine Auswahl an Werkzeugen zur Unterstützung bei der Validierung und Verifizierung. Enthalten sind unter anderem eine grafische Möglichkeit, Verkehrsszenarien aus einer Auswahl von vordefinierten Elementen zu erstellen, die Möglichkeit, Sensormodelle hinzuzufügen, und eine Schnittstelle für Steuersysteme.

↳ **AILiveSim:** Die Simulationsplattform von *AILiveSim* wurde für das Testen von künstlicher Intelligenz in einer Reihe von Bereichen wie Robotik, Logistik, Straßenverkehr, Schifffahrt und Computer Vision entwickelt. Sie verspricht dynamische virtuelle Umgebungen auf Basis der Unreal™ Engine, Tools für die Erstellung von Szenarien, hochauflösende Sensorsimulationen, Echtzeit-Datenanalyse und cloudbasierte Bereitstellung. Die Cloud-Unterstützung soll dabei skalierbare Simulationen ermöglichen, die die Entwicklung und Validierung von KI-basierten Systemen in der Industrie durch Einbindung in vorhandene kontinuierliche Integrationsprozesse weiter beschleunigen können. Ein Überblick über den generellen Aufbau der Plattform wird in [Leudet et al., 2019] gegeben.

↳ **BeamNG.tech[10]:** Die Simulationsplattform zeichnet sich durch einen leistungsstarken Soft-Body-Simulator, eine Vielzahl an Editoren sowie eine umfangreiche API aus. Letztgenannte ermöglicht die prozedurale Generierung von Inhalten, die automatische Sammlung von Simulationsdaten für Trai-

[6] https://www.dspace.com/de/gmb/home/products/sw/automotive_simulation_models.cfm [letzter Zugriff: 17.02.26]
[7] https://ipg-automotive.com/de/produkte-loesungen/software/carmaker/ [letzter Zugriff: 17.02.26]
[8] https://cosimate.com/ [letzter Zugriff: 17.02.26]
[9] https://plm.sw.siemens.com/de-DE/simcenter/autonomous-vehicle-solutions/prescan/ [letzter Zugriff: 17.02.26]
[10] https://beamng.tech/ [letzter Zugriff: 17.02.26]

ning und Validierung sowie die Steuerung von simulierten Fahrzeugen durch externe Programme. Ferner verfügt *BeamNG.tech* über eine große Menge nutzbarer virtueller Objekte. Darunter mehr als zwanzig Fahrzeuge, Hunderte Hindernisse und detaillierte Karten, welche eine Fläche von vierundvierzig Quadratkilometern abbilden, darunter städtische und vorstädtische Umgebungen. [Gambi et al., 2020] Die Software wird unter einer Kombination aus kommerziellen und Open-Source-Lizenzen angeboten. Die genannten Funktionen machen das System anpassungsfähig für Forschungs- und Entwicklungszwecke. Allerdings sind die zugrundeliegende Engine und ein Großteil der für diese Arbeit relevanten Softwarekomponenten nicht quelloffen.

↳ **Dutch Space BV EuroSim[11]: –** Das nicht quelloffene und kommerziell vertriebene Simulationswerkzeug *EuroSim* legt besonderen Wert auf Echtzeit-Simulationen mit einer hohen Genauigkeit. Entwickelt wurde es primär für den Einsatz im Bereich Luft- und Raumfahrt. Es eignet sich aber auch für den Einsatz in anderen Bereichen. Die Echtzeit-Simulationsfähigkeiten sind für Anwendungen von entscheidender Bedeutung, bei denen präzises Timing und Synchronisation erforderlich sind. Dies umfasst insbesondere Hardware-in-the-Loop (HiL)-Tests und missionskritische Simulationen. Ihre modulare Architektur verspricht die einfache Integration verschiedener Modelle und Komponenten und soll die Anpassung an unterschiedliche Projektanforderungen erleichtern. *EuroSim* wurde bereits vielfach erfolgreich im Rahmen von europäischen Raumfahrtmissionen eingesetzt. Die Zuverlässigkeit und Leistungsfähigkeit des Tools konnten so bereits in realen und missionskritischen Szenarien gezeigt werden.

5 FORSCHUNGSDESIGN

Die in Abschnitt 3 identifizierten und formulierten zentralen Forschungsfragen, welche am Ende dieser Arbeit beantwortet werden sollen, können in ihrer derzeitigen Form nicht ohne Weiteres eindeutig beantwortet werden. Dies ist zum einen darin begründet, dass sowohl die Leitfrage als auch die Teilfragen die Art und Weise adressieren, wie ein identifiziertes Problem gelöst werden kann:

„Wie kann […] modelliert werden, um […]?“
„Wie müssen […] aufgebaut sein, um […]?“
„Wie kann […] so unterstützt werden, dass […]?“
„Wie kann ermöglicht werden, dass […]?“

Eine derart formulierte Frage lässt viel Handlungsspielraum offen, da mit sehr hoher Wahrscheinlichkeit nicht die eine einzigartige Lösung existiert, die diese Frage nach dem *Wie* beantworten könnte. Die Forschungsfrage und deren Teilfragen können aus mathematischer Sichtweise daher als sogenanntes *inkorrekt gestelltes Problem* (engl.: ill-posed problem) betrachtet werden, da es zwar höchstwahrscheinlich eine Lösung gibt (sonst würde die vorliegende Arbeit nicht existieren), aber die Lösung ebenso wahrscheinlich nicht einzigartig (engl.: unique) zu sein scheint [Loehle, 2011; Yuri P. & Sizikov, 2005]. Da die identifizierten Probleme (vgl. Abschnitte 1 und 2) bei der Entwicklung von softwarebasierten Systemen, als Schritt auf dem angestrebten Weg zum autonomen Schiff, aber potenziell substanzielle Hinderungen darstellen, ist die Ergründung und Beantwortung der gestellten Fragen aufgrund des hohen Erkenntnisinteresses trotzdem wissenschaftlich erstrebenswert.

[11] `https://eurosim.nl/` [letzter Zugriff: 23.08.24]

5.1 EXPLORATIVE UND KONSTRUKTIVE PHASE

Um die Forschungsfragen trotz der genannten Unschärfe beantworten zu können, ist diese Arbeit in zwei Forschungsphasen aufgeteilt. Zunächst werden die in Teil I aufgestellten Forschungsfragen herangezogen, um sie in Teil II als Ausgangspunkt einer explorativen ersten Phase zu nutzen. Explorative Untersuchungen zielen unter anderem darauf ab, in relativ unerforschten oder nicht unmittelbar klar abgrenzbaren Untersuchungsbereichen theoretische und begriffliche Voraussetzungen zu schaffen, um erste Hypothesen formulieren zu können [Bortz & Döring, 2002]. Es wird also das in den bisherigen Abschnitten abgesteckte Forschungsfeld empirisch (d. h. aus bisheriger Erfahrung gewonnen) erkundet, Daten werden gesammelt und der Versuch wird unternommen, Auffälligkeiten zu erkennen, ohne dabei einer bestimmten Hypothese nachzugehen. Die explorative Phase wird im vorliegenden Fall genutzt, um den aktuellen Stand der Wissenschaft und Technik bezüglich des szenariobasierten Einsatzes von Simulationen im Rahmen der Verifikation und Validierung von softwarebasierten automatisierten maritimen Fahrsystemen systematisch zu erfassen. Auf Basis der gewonnenen Übersicht über die relevanten Themenfelder und ihre Hintergründe werden anschließend Anforderungen an einen solchen Einsatz von Simulationsanwendungen aggregierend formuliert. Anhand dieser werden dann die konkreten aktuellen Forschungslücken identifiziert.

Die gestellten Anforderungen in Verbindung mit den identifizierten Lücken und Schwachpunkten werden nachfolgend in Handlungsempfehlungen für das weitere Vorgehen zusammengefasst, aus welchen sich gleichzeitig konkrete gestaltungsorientierte Ziele für den praktischen Teil dieser Arbeit ableiten lassen. Positiv hervorgehobene bestehende methodische und technologische Ansätze werden zudem als direkter Input für die Konzeptentwicklung in dieser zweiten Phase der vorliegenden Arbeit genutzt. Durch die logisch kohärente Beziehung, der abgeleiteten gestaltungsorientierten Ziele zu den Forschungsfragen und den Handlungsempfehlungen, kann anschließend die Hypothese aufgestellt werden, dass ein Artefakt, das die aufgestellten Ziele erfüllt, in der Lage ist, die im Rahmen der Problembeschreibung identifizierten Herausforderungen zu bewältigen, genauer gesagt bei deren Bewältigung dienlich sein kann, und infolgedessen eine Antwort auf die Forschungsfragen darstellt. Die entsprechend in Abschnitt 12 formulierte Hypothese dient auf Basis dieser Eigenschaften gemeinsam mit den in gestaltungsorientierten Zielen als Schlüsselelement dieser Forschungsarbeit, da sie die Ergebnisse der konstruktivistischen Phase mit den eingangs aufgestellten Forschungsfragen verknüpft. Der Zusammenhang der forschungsprozessrelevanten Artefakte ist vereinfacht in Abbildung 3 dargestellt.

Aufbauend auf der in Teil II geschaffenen Basis kann konstruktionswissenschaftlich [Wilde & Hess, 2006] an der Erreichung der definierten gestaltungsorientierten Ziele gearbeitet werden. Konstruktionswissenschaftliche Methoden streben nach Erkenntnisgewinn durch Schaffen und Evaluieren von informationstechnischen Lösungen in Form von Modellen, Methoden und Systemen [Simon, 1996]. Da diese Arbeit die beiden eng miteinander verbundenen, aber nicht kongruenten Teilbereiche *Simulation* und *Szenarien* betrachtet, kommen zwei verbreitete und hier aufeinander aufbauende Methoden zum Einsatz: *Modellierung* und *Entwicklung und Test von Prototypen* [Wilde & Hess, 2006; Budde et al., 1992]. Die Konzepte werden dabei genau auf die in der explorativen Phase identifizierten Lücken der aktuellen Praxis ausgerichtet, indem die aufgestellten Handlungsempfehlungen gezielt angegangen werden.

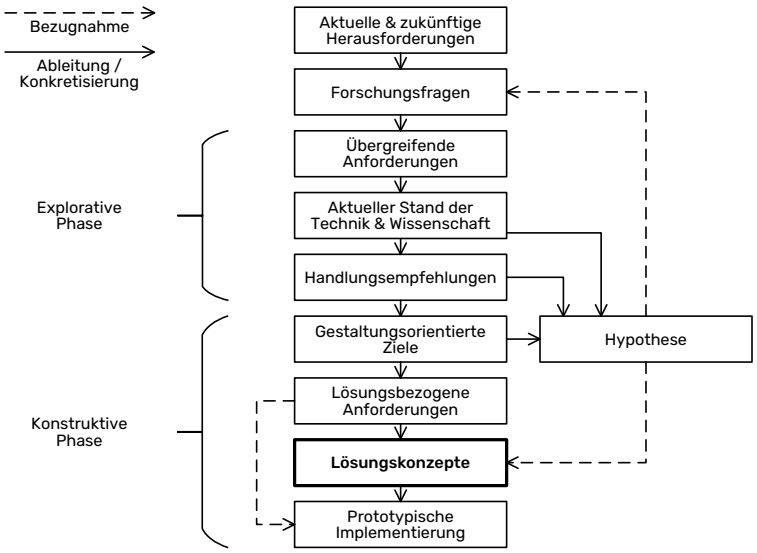


Abbildung 3: Die relevantesten im Rahmen dieser Arbeit erzeugten Artefakte. Erkennbar sind die stringenten Zusammenhänge. Die Hypothese fungiert als Bindeglied zwischen den beiden Phasen dieser gestaltungsorientierten Forschungsarbeit und stellt die argumentative Grundlage zur Beantwortung der Forschungsfragen durch die Implementierung dar.

Durch eine prototypische Umsetzung der erarbeiteten Konzepte können anschließend die zentrale Forschungsfrage und ihre Teilfragen beantwortet werden, bzw. der Kreis der möglichen Antworten eingegrenzt werden, in dem der geschaffene Prototyp mit den lösungsbezogenen Anforderungen, welche auf den Handlungsempfehlungen basieren, abgeglichen wird und somit die übergeordneten Anforderungen an den generellen Einsatz von Simulationsanwendungen zu szenariobasierten Testzwecken im Rahmen von V&V überprüft werden. Bei Erfüllung der lösungsorientierten Anforderungen ist durch diese linear-kausale Verknüpfung gegeben, dass die aufgestellte Hypothese als verifiziert angesehen werden kann und somit das Artefakt und die diesem zugrunde liegenden erstellten (technischen) Konzepte als Antworten auf die Forschungsfragen dienen können. Diese Beziehungen zwischen den wesentlichen Ergebnissen sowie die daraus für diese Arbeit abgeleiteten argumentativen Zusammenhänge sind in Abbildung 3 dargestellt. Da die Zielsetzung eines experimentellen Prototyps die Untersuchung der technischen Machbarkeit zuvor geschaffener neuer Konzepte und nicht die Auslieferung eines betriebsbereiten Produktes ist [Budde et al., 1992], sind dabei die erarbeiteten Konzepte als der Hauptbeitrag dieser Arbeit zu betrachten und der Prototyp selbst lediglich als der Beweis der technischen Umsetzbarkeit dieser.

5.2 AUFBAU UND METHODENAUSWAHL

Um ein zielgerichtetes Vorgehen zu garantieren, ist ein systematisches Vorgehen bei der Bearbeitung wissenschaftlicher Fragestellungen unabdingbar. Damit das zuvor beschriebene Vorgehen erfolgreich sein kann, orientieren sich die beiden Studienphasen daher jeweils an einem für die jeweilige Phase geeignetem, verbreitetem und erprobtem Vorgehensmodell.

Explorative Phase

Die explorative Phase dieser Arbeit zielt darauf ab, neue Erkenntnisse, Sachverhalte und Zusammenhänge aus existierenden Erfahrungen und Ergebnissen beschreibend (d. h. deskriptiv, [Kromrey, 1994]) abzuleiten. Um dieser Aufgabe gerecht zu werden, orientiert sie sich an dem allgemeinen Vorgehensmodell empirischer Forschung, welches den Ablauf eines Erkenntnisprozesses in die sechs Phasen Er-

kundungsphase, theoretische Phase, Planungsphase, Untersuchungsphase, Auswertungsphase und Entscheidungsphase unterteilt [Bortz & Schuster, 2010; Bortz, 1979].

In der Erkundungsphase geht es zunächst darum, sich selbst ein Bild über mögliche Zusammenhänge zu verschaffen, um diese danach in der Theoriephase zu einem konsistenten Modell zu verknüpfen [Cleff, 2015]. Zu diesem Zweck werden in den Abschnitten 6 und 7 zunächst anhand einschlägiger Literatur [Bortz & Schuster, 2010] die methodischen und technologischen Grundlagen für das betrachtete Gebiet erarbeitet. Methodisch als besonders relevant hervorzuheben sind hierbei das Methodenspektrum der Verifizierung und Validierung (V&V) sowie die Absicherung von Systemen durch szenariobasiertes Testen. Technologisch werden insbesondere moderne automatisierte maritime Fahrsysteme und ihre Entwicklung, Simulationsanwendungen in verschiedenen Ausprägungen und die Modellierung von Szenarien als Verbindungsstück zwischen diesen beiden genauer betrachtet. Basierend auf den so erschlossenen Grundlagen werden Anforderungen an Simulationsanwendungen und -modelle bezüglich der optimalen Unterstützung des V&V-Prozesses im Rahmen der szenariobasierten Absicherung von automatisierten maritimen Fahrsystemen definiert.

Abschnitt 10.1 stellt die Planungsphase des empirischen Vorgehensmodells dar. Es wird die konkrete Erhebungsmethode beschrieben und deren Parameter werden festgelegt. Da es das Ziel der Vorstudie ist, den aktuellen Umfang des Einsatzes von szenariobasierten Simulationen im Rahmen von V&V maritimer Fahrsysteme im Hinblick auf die zuvor definierten Anforderungen zu untersuchen, wäre es im Rahmen eines empirischen Vorgehens üblich, Nutzer von eben solchen zu befragen. Da ebendiese Nutzer aber nicht sehr zahlreich sind und dazu weltweit verteilt, scheint eine solche Befragung nicht praktikabel umsetzbar, zumindest nicht ohne die Gefahr der Gewinnung von verzerrten Studienergebnissen. Alternativ wird ein systematisches Literaturreview genutzt, um das benötigte Wissen zu gewinnen. Dieses Vorgehen ist in den meisten Fachrichtungen zwar seltener anzutreffen, im Fachgebiet der Informatik aber nicht gänzlich unüblich [Raghupathi et al., 2022; Ramesh et al., 2004]. Um dem Anspruch der Empirie trotzdem gerecht zu werden, wird das Literaturreview nicht unmittelbar darauf ausgerichtet, Arbeiten über Simulationsanwendungen selbst zu identifizieren und zu analysieren. Stattdessen werden durch eine gezielte Parametrierung des Vorgehens, Arbeiten adressiert, welche die Forschung und Entwicklung von automatisierten maritimen Fahr- und Assistenzsystemen als zentralen Gegenstand haben und im Rahmen dessen eine Simulationsanwendung entwicklungsunterstützend oder zur V&V im Rahmen der Evaluation einsetzen. Somit ist der primäre Forschungsgegenstand die individuelle Art und Weise, wie Simulationen von wissenschaftlichen Nutzern eingesetzt werden, und die Anforderung an eine empirische Studie ist gegeben.

Die Untersuchungsphase umfasst daher die Durchführung der zuvor geplanten Datenerhebung und die aggregierende Auswertung der erhobenen Daten in Form eines systematischen Literaturreviews. Zur Wahrung der wissenschaftlich essenziellen Nachvollziehbarkeit sind im Abschnitt 10.2 die Zwischenergebnisse der einzelnen Prozessschritte detailliert festgehalten. In der Auswertungsphase werden anschließend die zuvor gewonnenen und aggregierten Daten verarbeitet und bewertet [Bortz & Schuster, 2010]. Die Ergebnisse der Auswertung sind in Abschnitt 10.3 in einer den zu Beginn festgelegten Anforderungen gegenüberstellenden Form festgehalten.

Abschließend werden auf Basis der aufgedeckten Lücken und Chancen in der bisherigen Anwendung von Simulationsanwendung im Kontext der szenariobasierten Absicherung maritimer softwarebasier-

ter Fahrsysteme Handlungsbedarfe aufgelistet. Diese sind so formuliert, dass sie eine Schließung der Lücken bei gleichzeitigem Nutzen der Chancen adressieren. Diese stellen die Basis für die konstruktive Phase dieser Arbeit dar und der entsprechende Abschnitt 11 bildet somit die Entscheidungsphase, die im Rahmen der empirischen Forschung den zukünftigen Einsatz der Ergebnisse zum Inhalt hat [Lamm, 2023; Cleff, 2015].

Konstruktive Phase

Auch in der zweiten Forschungsphase dieser Arbeit ist ein systematisches Vorgehen nötig. Vor dem Hintergrund, dass sich spätestens aus der Nutzung der konstruktionswissenschaftlichen Forschungsmethode des Prototyping die Notwendigkeit der Entwicklung einer softwaretechnischen Simulationsanwendung für maritimen Verkehr ergibt, orientiert sich diese Arbeit zusätzlich an den typischen Entwicklungsphasen der Softwaretechnik. Am geläufigsten ist dabei wohl die Einteilung des Software-Lebenszyklus in die Phasen der Spezifikation, des Entwurfs, der Implementierung, der Installation sowie des Betriebs eines Softwaresystems [Balzert, 2011; Partsch, 2010].

Ausgehend von den in der explorativen ersten Phase dieser Arbeit formulierten Handlungsempfehlungen werden in Abschnitt 12 konkrete gestaltungsorientierte Ziele für die zu erschaffende Lösung formuliert. Anschließend werden in Abschnitt 13 die konkreten und eindeutigen lösungsbezogenen Anforderungen an die zu erzeugenden softwaretechnischen Artefakte formuliert – das Ergebnis ist eine Spezifikation [Balzert, 2011] im Sinne des Software-Lebenszyklus. Diese Phase bezieht gewonnene Erkenntnisse aus allen vorangegangenen Abschnitten mit ein.

Balzert [2011] definiert die Entwurfsphase wie folgt: „Ausgehend von der Spezifikation ist es die Aufgabe im Entwurf, die Software-Architektur zu erstellen. Beim Entwurf wird oft noch zwischen ‚Entwurf im Großen' und ‚Entwurf im Kleinen' unterschieden." Entsprechend orientiert sich der Abschnitt 14 ebenfalls an dieser stufenweisen Verschiebung der Betrachtungsebene vom Abstrakten zum Spezifischen: In Abschnitt 14.1 werden die übergreifenden Konzepte erarbeitet, welche in Abschnitt 14.3 auf einzelne zentrale Bestandteile aufgebrochen und im Detail ausgearbeitet werden.

Abschnitt 15 beschreibt die experimentelle Umsetzung der Konzepte in Form eines Prototyps zur Überprüfung der technischen Machbarkeit der erarbeiteten Konzepte. Da das Konzept bis zu diesem Punkt größtenteils technologieagnostisch erarbeitet wurde, wurden zunächst noch technische Details ausgestaltet und konkrete geeignete Technologien für die Umsetzung identifiziert. Anschließend werden in den Abschnitten 15.1, 15.2 und 15.3 die zentralen Bestandteile in Hinblick auf ihre technische Umsetzung genauer betrachtet.

Laut [Partsch, 2010] hat die vorletzte Phase den Test und die Abnahme inne. Beides manifestiert sich in dieser Arbeit in der Evaluation des Prototyps in Hinblick auf die Erreichung der in Abschnitt 12 formulierten Ziele und somit, wie zuvor dargestellt, auch in Hinblick auf die Eignung zur Beantwortung der in Abschnitt 3 aufgestellten Forschungsfragen. Die Evaluation ist Inhalt der Abschnitte 16.1 bis 16.2.1. Die letzte Phase des Software-Lebenszyklus, der Produktiveinsatz der Software, bestehend aus Betrieb und Wartung, ist nicht mehr Teil dieser Arbeit, da die erzeugten Softwareartefakte im Sinne der Forschungsmethode des experimentellen Prototyping nicht intendieren, die Ansprüche zu erfüllen, die an betriebsbereite Software im Sinne von Produktiveinsätzen gestellt werden.

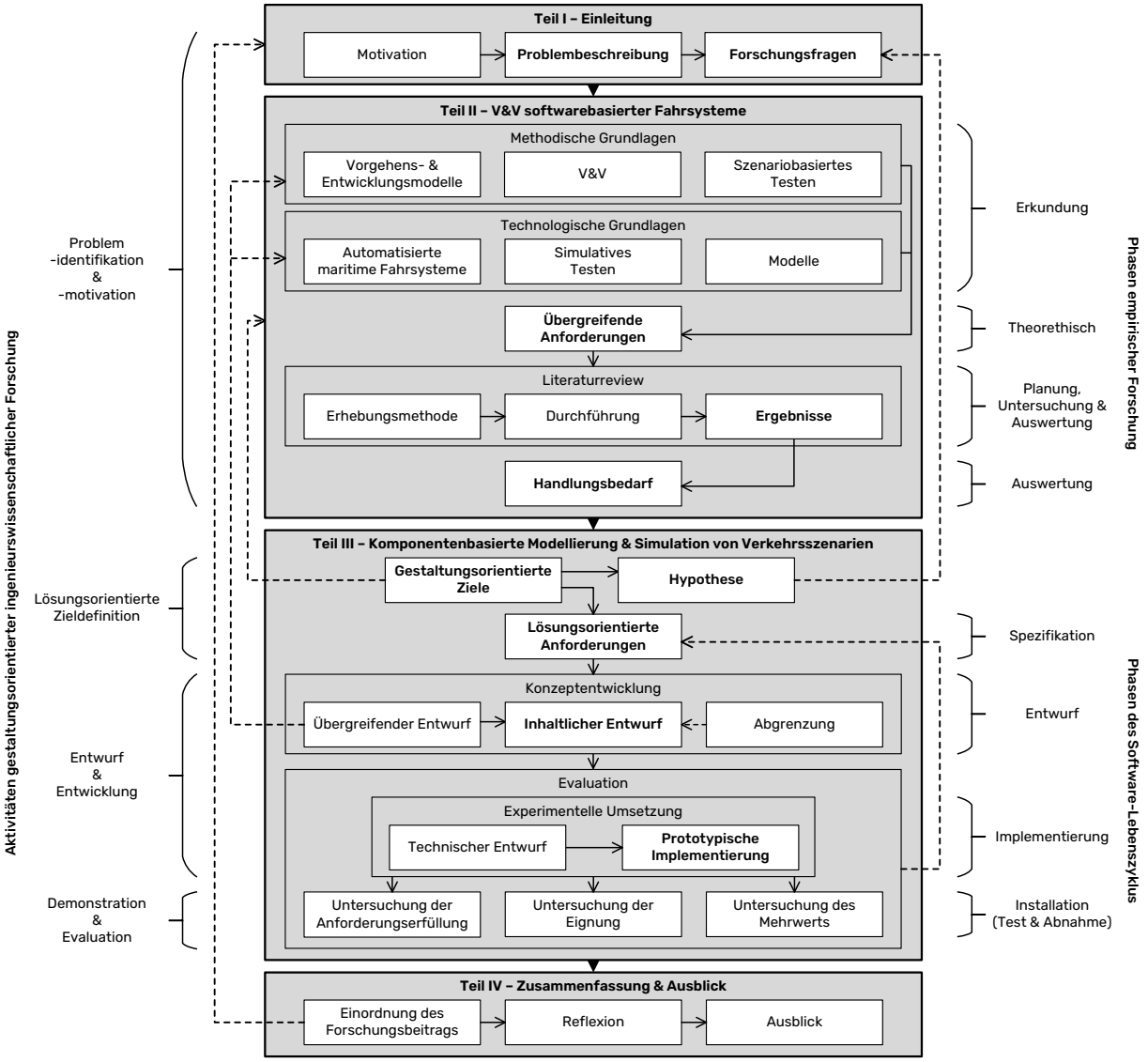


Abbildung 4: Aufbau der vorliegenden Arbeit nach ingenieurswissenschaftlicher Methodik. Zusätzlich zu den aufeinander aufbauenden Kapitelinhalten sind Informationsflüsse und inhaltlichen Rückbezüge dargestellt.

Zusammenfassend lässt sich die Gesamtstruktur der vorliegenden Arbeit der designwissenschaftlichen Forschung zuordnen, welche im Deutschen üblicherweise als gestaltungsorientierte ingenieurswissenschaftliche Forschung bezeichnet wird. Das Ziel gestaltungsorientierter Forschung besteht darin, Wissen und Verständnis für ein Problem und dessen Lösung durch die Planung, Erstellung und den Einsatz eines Artefakts zu gewinnen. Der Begriff *Artefakt* beschreibt etwas, das von Menschen konstruiert wurde, im Gegensatz zu etwas natürlich Vorkommendem. Solche Artefakte müssen die bestehenden Lösungen für ein Problem verbessern oder zumindest eine erste Teillösung für ein vielfältiges Problem bieten. Informationstechnische Artefakte lassen sich im Allgemeinen einer der folgenden Gruppen zuordnen: Konstrukte (Vokabular und Symbole), Modelle (Abstraktionen und Darstellungen), Methoden (Algorithmen, Verfahren und Konzepte), Instanziierungen (vollständig implementierte und prototypische Systeme), Entwurfstheorien. [Hevner & Chatterjee, 2010; Simon, 1996]

Die sechs aufeinanderfolgenden Aktivitäten dieser Methodik (vgl. [Hevner & Chatterjee, 2010]) sind zusätzlich in Abbildung 4 auf der linken Seite einordnend dargestellt. Die zuvor beschriebenen Vorgehensmodelle lassen sich jeweils einem oder mehreren dieser Schritte zuordnen und stehen nicht im

Widerspruch mit der Gesamtmethodik oder den einzelnen Schritten, da der Erfolg der designwissenschaftlichen Methodik wesentlich von der Auswahl erprobter und geeigneter Methoden für die einzelnen Aktivitäten abhängt. [Hevner & Chatterjee, 2010]

Zu guter Letzt werden die Ergebnisse der vorliegenden Arbeit in Abschnitt 17 zusammengefasst und anschließend wird noch einmal objektiv und kritisch reflektierend auf den durchgeführten Forschungsprozess zurückgeblickt. Abschließend werden zudem mögliche Erweiterungen und Verbesserungen beleuchtet und potenzielle Ansätze knapp skizziert.

II

Verifizierung & Validierung softwarebasierter maritimer Fahrsysteme

Das angestrebte Ergebnis dieser Arbeit ist die ganzheitliche Unterstützung der Entwicklung und der Abnahme softwarebasierter automatisierter Systeme moderner maritimer Fahrzeuge. In Teil I wurde festgestellt, dass für diese Aufgabe gezielte, szenariobasierte, simulative Testläufe das größte Potenzial bieten. Das Ergebnis dieser Arbeit muss somit ein softwaretechnisches Artefakt sein. Um dieser ingenieurswissenschaftlichen Aufgabe gerecht zu werden, muss zunächst ein klares Bild des Anwendungsgebietes der angestrebten Ergebnisse sowie der technologischen Besonderheiten innerhalb dessen bestehen [Hevner et al., 2004]. Daher müssen die relevanten Technologien und Ansätze, die den aktuellen Stand der Technik darstellen, vorab aufgearbeitet werden. Im Folgenden werden die drei Kernthemen näher betrachtet, aus denen sich simulations- und szenariobasierte Validierung und Verifizierung moderner softwarebasierter maritimer Systeme zusammensetzt, um ein klares Verständnis des Themenkomplexes für die restlichen Teile der vorliegenden Arbeit zu schaffen: Abschnitt 6 befasst sich mit modernen maritimen Fahrzeugen, der Entwicklung softwarebasierter Systeme für solche und den üblicherweise eingesetzten Entwicklungsmethoden, Abschnitt 7 beleuchtet Methoden zum Beweis der Sicherheit solcher Systeme und stellt die Grundlagen zum Einsatz von Simulationen als Testwerkzeuge dar. Da andere Verkehrsdomänen mitunter in diesen Bereichen bereits weiter fortgeschritten sind, wird in Abschnitt 8 zusätzlich ein Blick über den Tellerrand geworfen und die gängige Praxis in Bezug auf den Einsatz von Simulationen im Bereich der Straßen- und Luftfahrzeuge untersucht. Abschließend werden in Abschnitt 9 auf Basis des so gewonnenen Wissens Anforderungen für den zukünftigen entwicklungsbegleitenden szenariobasierten Einsatz von Simulation aufgestellt.

Wie in Teil I bereits besprochen wurde, ist es in dem vorliegenden Fall zusätzlich notwendig, das Forschungsgebiet selbst aufzuarbeiten und die bestehenden Herausforderungen zu konkretisieren. Zur Erledigung dieser Aufgabe wurde ein systematisches Literatur Review durchgeführt. Ein systematisches Literaturreview zielt auf eine Zusammenfassung eines Forschungsgebietes ab, die bei der Identifizierung möglicher spezifischer Forschungsfragen hilfreich ist. [Rowley & Slack, 2004]. Um dieses Ziel zu erreichen, sind ein zielgerichtetes Vorgehen und eine gute Vorbereitung der eigentlichen Literatursuche und -auswertung notwendig. Unabdingbar dafür ist eine genaue Vorstellung davon, wo die Grenzen der Betrachtung liegen. Daher werden in Abschnitt 10.1 zunächst die verwendete Erhebungsmethode beschrieben und die verwendeten Parameter detailliert dargestellt. Anschließend wird die Durchführung der Literatursuche nachvollziehbar dargestellt, bevor die resultierenden Ergebnisse mit den zuvor festgelegten Anforderungen abgeglichen werden. Abschließend werden in Abschnitt 11 die durch diesen Abgleich extrahierten, bisher nicht klar bekannten Lücken in der aktuellen Anwendung von Simulationen und Szenarien zur Verifizierung und Validierung maritimer automatisierter Systeme in Form von Handlungsbedarfen zusammengefasst. Diese Handlungsbedarfe konkretisieren den Weg, den es zur Beantwortung der aufgestellten Forschungsfragen zu beschreiten gilt, und bilden somit die Basis für den nachfolgenden lösungsbezogen konstruktiven Teil dieser Arbeit.

6 ENTWICKLUNG MODERNER SEESCHIFFE

Die maritime Transportindustrie steht aktuell an der Schwelle zu einer technologischen Revolution, die den Entwurf, den Betrieb und das Management von Schiffen verändern wird. Im Mittelpunkt dieses Transformationsprozesses stehen Fortschritte in der Automatisierung von Schiffen und im Einsatz von künstlicher Intelligenz im Rahmen dessen [Askari & Hossain, 2022; Wright, 2020]. Die Entwicklung hochautomatisierter und autonomer Seeschiffe stellt einen signifikanten Fortschritt in diesem Kontext dar und verspricht, die Effizienz, Sicherheit und Nachhaltigkeit in der globalen Schifffahrt zu verbessern [Askari & Hossain, 2022; Wright, 2020; Munim, 2019; Burmeister et al., 2014]. Mit der fortschreitenden Automatisierung von Schiffen gewinnen die Konzeption, Implementation und Integration von robusten und sicheren softwarebasierten Steuerungssystemen und Entscheidungsalgorithmen zunehmend an Bedeutung.

Schiffe unterscheiden sich von vielen Fahrzeugen in anderen Verkehrsdomänen wie dem Straßenverkehr in zwei wichtigen Aspekten: (1) Schiffe selbst sind große, langlebige und hochwertige Güter. Die Eigner wollen daher in der Regel ein hohes Maß an Kontrolle über sie behalten. Ferner setzen aktuelle internationale Rechtsvorschriften die Anwesenheit einer qualifizierten Person voraus, in der Regel des Kapitäns, der für die Überwachung des Betriebs verantwortlich ist. (2) Für viele Arten von Schiffen ergeben sich die größten Vorteile des Übergangs zum autonomen Betrieb daraus, dass kein Personal an Bord ist. Denn so entfällt der Bedarf für Unterkunftsbereiche und die meisten Rettungsmittel, was die Entwicklung neuer Schiffs- und Transportsystemkonzepte ermöglicht, die das Potenzial haben, die zukünftige Entwicklung des Seeverkehrs erheblich beeinflussen zu können. Daher ist anzunehmen, dass der vollständig unbemannte Betrieb ein vorrangiges Ziel der Forschungs- und Entwicklungsarbeiten im Zusammenhang mit modernen Schiffen sein wird. [Rødseth & Nordahl, 2018]

Dieser Abschnitt widmet sich daher speziell den technischen, technologischen und regulatorischen Grundlagen moderner Seeschiffe, der dort verbauten Systeme und ihrer Besonderheiten. So werden ergänzend zu den Abschnitten 1 und 2 die Relevanz der vorliegenden Arbeit noch einmal verdeutlicht und die Grundlage für im weiteren Verlauf der Arbeit getroffene Entscheidungen geschaffen.

6.1 AUTOMATISIERUNG UND AUTONOMIE

Die maritime Industrie, allen voran die Seeschifffahrt – also die gewerbsmäßige Beförderung von Gütern über die Meere und Ozeane dieser Welt – hat globalwirtschaftlich betrachtet einen hohen Stellenwert für die moderne Gesellschaft, in der wir leben. So ist sie für den Transport von Gütern im Wert von mehreren Milliarden US-Dollar täglich verantwortlich [Walker et al., 2019] und befördert zusammengefasst regelmäßig über 90 % des globalen Handelsvolumens, betrachtet man ausschließlich das Gewicht der transportierten Güter über den Zeitraum von einem Jahr [Walker et al., 2019]. Und obwohl der Transport über den Seeweg deutlich zeitaufwendiger ist als z. B. der Transport über den Luftweg, ist die Erklärung in unserer kapitalistisch geprägten Welt leicht zu finden: Unter den verfügbaren Transportmitteln ist die Seeschifffahrt die effizienteste Art des Langstreckentransports großer Mengen von Gütern [Rodrigue, 2013] und infolgedessen die preiswerteste: Stetig größer werdende Schiffe können immer mehr Volumen laden und die Betriebskosten pro Transporteinheit bleiben deutlich unter denen der Alternativen. So konnte sich die industrielle Seeschifffahrt als Rückgrat unse-

rer global vernetzten Wirtschaft etablieren, deren zukünftige Entwicklung wohl bis auf wenige Ausnahmen jeden Menschen in der einen oder anderen Weise betrifft [Kaluza et al., 2010].

Dass Bemühungen unternommen werden, die industrielle Seeschifffahrt noch effizienter, aber auch sicherer und nachhaltiger zu machen, ist daher wenig überraschend. Im Fokus dieser Forschungs- und Entwicklungsbemühungen stehen meist die für die globale Wirtschaft relevantesten Schiffstypen: Massengutfrachter, Tanker und Containerschiffe. Diese haben gemeinsam, dass sie große Mengen transportieren können. Mittlerweile sind Schiffe dieser Typen derart ausladend geworden, dass sie ohne elektronische Unterstützung nur noch schwer oder gar nicht mehr zu manövrieren wären. Entsprechend hat die Nachfrage nach Automatisierung signifikant zugenommen. Insbesondere die Forschung in den Bereichen automatisierte Navigation und Kollisionsvermeidung erfährt im Kontext des Seeverkehrs aktuell eine zunehmende Relevanz [Raza et al., 2022; Wright, 2020]. Das Ziel dieser Entwicklungen sind autonome Varianten der genannten industriell relevanten Transportschiffstypen.

Auf dem Weg zur Erreichung dieses Ziels werden die nötigen Technologien naturgemäß zunächst im Kleinen entwickelt und erprobt. Ein solches Projekt war der 2019 ins Leben gerufene *Clearbot*[12], welcher in Indonesien als Studierendenprojekt zur Unterstützung bei der effizienten Reinigung der Wasserwege begann. Zuvor waren einfache Ruderboote und Netze noch die gängigste Lösung in der Region, sodass die Einheimischen nicht in der Lage waren, den Müllmassen Herr zu werden. Nachdem die Beteiligten die Mehrwerte der verwendeten Technologien, wie reduzierte Kosten und reduzierten Bedarf an Personal, erkannt hatten, wurden die Bemühungen auf weitere Anwendungsbereiche, wie die Lieferung von kleinen Gütern oder die Reinigung von Wasseraufbereitungsanlagen, erweitert.

Ebenfalls 2019 erhielt das koreanische Unternehmen *Daewoo Shipbuilding & Marine Engineering*[13] die grundsätzliche Genehmigung (engl.: Approval in Principle, AIP) für seine Plattform für „intelligente" Schiffe *DSME Smartship Solutions* (DS4). Drei Jahre später wurde die Plattform auf dem Demonstrationsboot *DAN-V* integriert und bei einer knapp 30 km weiten Testfahrt vor der Küste von Seoul im kleinen Maßstab erprobt[14]. Dabei war kein Personal an Bord des Bootes. Stattdessen wurden Befehle von einer landseitigen Kontrollstation aus an die DAN-V übertragen. Das Boot fuhr auch automatisiert entlang einer geplanten Route und wich dabei wahrscheinlichen Kollisionen aus.

Aber auch einzelne Funktionen größerer Schiffe wurden bereits erfolgreich automatisiert und befinden sich teilweise schon im Einsatz. Das australische Unternehmen *MAID Systems* vertreibt das *Marine Autonomous Intelligent Docking*[15] System, das laut eigener Aussage das weltweit erste und einzige vollständig autonome Andocksystem für Schiffe ist. Die Integration von Foto- und Infrarotsensoren in das Schiff ermöglicht die Echtzeit-Erfassung der Umgebung in einem Umkreis von mehreren hundert Metern. Mit Hilfe dieser Sensoren ermöglicht das System dem Schiffsführer die vollständig automatisierte Neupositionierung des Schiffes nach Auswahl einer Zielposition über eine grafische Benutzerschnittstelle (engl.: Graphical User Interface, GUI). In der Folge navigiert das Schiff von seiner aktuellen Position zu der angegebenen Position. Diese Position kann sich beispielsweise an einem Dock oder Objekt befinden.

[12] `https://clearbot.org/` [letzter Zugriff: 17.02.26]
[13] Nach der Übernahme durch den Konzern *Hanwha* wurde das Unternehmen in *Hanwha Ocean* umbenannt.
[14] `https://safety4sea.com/dsme-carries-out-autonomous-navigation-trials-with-remote-controls` [letzter Zugriff: 17.02.26]
[15] `https://maidsystems.com/ip-applications.php` [letzter Zugriff: 17.02.26]

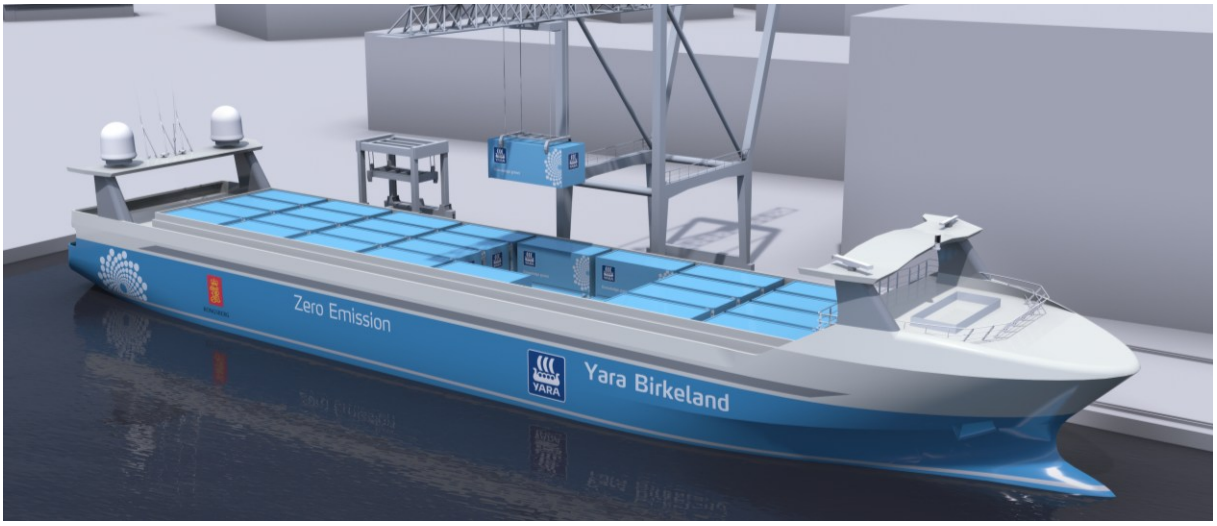


Abbildung 5: Computergrafische 3D Illustration des Containerschiffes Yara Birkeland, welches zukünftig vollständig automatisiert und ohne Personal an Bord zwischen dem Herøya-Industriepark und den Häfen Brevik und Larvik verkehren soll. [Yara International ASA, 2022]

Des Weiteren werden auch bereits größere Schiffe mit vollumfänglicher Automatisierung aller Systeme zur Steuerung getestet. Die *Yara Birkeland*[16] (vgl. Abbildung 5) gilt als das erste vollelektrische und autonome Containerschiff der Welt. Das mit einer Kapazität von 120 TEU verhältnismäßig kleine Containerschiff wurde in Zusammenarbeit von dem Chemieunternehmen *Yara International* und der *Kongsberg* Gruppe entwickelt und stellt einen Meilenstein beim Übergang der maritimen Industrie zu einem nachhaltigeren Betrieb dar. Das Schiff wurde mit dem Ziel konzipiert, die Emissionen beim Transport von Kunstdünger zwischen der Düngemittelfabrik von *Yara* in Porsgrunn und den Häfen von Brevik und Larvik zu reduzieren. Dadurch können jährlich bis zu 40.000 Lkw-Fahrten eingespart werden. Diese Umstellung trägt zu einer spürbaren Verringerung der CO_2-Emissionen und der Auslastung des Straßennetzes bei. Die *Yara Birkeland* wurde dabei von Beginn an für den autonomen Betrieb ohne Besatzung ausgelegt und mit moderner Technologie ausgestattet, darunter hochentwickelte Sensorsysteme für Navigation und Kollisionsvermeidung. Seit dem Jahr 2022 befindet sie sich allerdings in einer Testphase, in welcher weiterhin Personal an Bord ist, welche voraussichtlich im Jahr 2026 beendet werden wird. Danach soll das Schiff seine Reisen allein antreten, aber weiterhin von landseitigen Stationen beobachtet werden, damit im Notfall eingegriffen werden kann.

Anhand der genannten Beispiele ist zu erkennen, dass aktuell unterschiedlichste Systeme zur Automatisierung von Schiffen entwickelt und getestet werden. Diese unterscheiden sich nicht nur in ihrer Architektur, der Einbindung in das Schiff und der Funktionalität, sondern vor allem in dem Grad, zu dem das Schiff durch ihren Einsatz automatisiert wird. Diese unterschiedlichen Betrachtungen haben zu dem Aufkommen von einer Vielzahl an Bezeichnungen für automatisierte Schiffe unterschiedlicher Automatisierungsgrade geführt: Beinahe jedes Projekt definiert ein automatisiertes Schiff anders [Raza et al., 2022; Wright, 2020; Vagia et al., 2016]. Als Oberbegriffe haben sich an vielen Stellen Begriffe wie *Autonomous Surface Vehicle (ASV) [Kufoalor et al., 2019], Autonomous Maritime Surface Vessel (AMSV) [Raza et al., 2022]* und *Maritime Autonomous Surface Ships (MASS) [IMO, 2022]* etabliert. Diese drei Begriffe beinhalten alle den Begriff der Autonomie, welcher vom altgriechischen αὐτονομία (autonomía) abstammt und Eigengesetzlichkeit/Selbstständigkeit bedeutet. Das *Digitale Wörterbuch*

[16] https://yara.com/news-and-media/media-library/press-kits/yara-birkeland-press-kit/ [letzter Zugriff: 17.02.26]

der deutschen Sprache (DWDS) definiert Autonomie als „(Willens-)Freiheit einer Person, Institution o. Ä., über die sie betreffenden wesentlichen Sachverhalte selbst entscheiden (und entsprechend handeln) zu können“[17]. Autonomie kommt ingenieurwissenschaftlich betrachtet somit im eigentlichen Wortsinn einer vollständigen Automatisierung gleich, bei der keine Menschen involviert sind, die direkte Befehle geben oder im Notfall eingreifen könnten [Rødseth & Vagia, 2020; Wright, 2020; Albus & Antsaklis, 1998]. Dabei ist es nicht relevant, ob diese sich an Bord des Schiffes selbst befinden oder mittels digitaler Kommunikation von Land aus handeln. Die Eigenschaft der Autonomie wird von den oben genannten Projekten daher bisher lediglich von *Clearbot* erfüllt. Die Begriffe ASV, AMSV und MASS scheinen somit erst einmal irreführend und bedürfen somit einer Einordnung, weiteren Unterteilung und Differenzierung. Dabei sei auch ein Blick in andere Verkehrsdomänen gewagt: Unter anderem aufgrund der früher intensivierten Entwicklung automatisierter Fahrzeuge für den Einsatz im öffentlichen Raum und des damit einhergehenden breiten öffentlichen Interesses liegen hier bereits offizielle Normen und Standards vor. So hat die gemeinnützige Organisation *SAE International*[18] bereits im Jahr 2014 [SAE, J3016:JAN2014] die Norm *SAE J3016* veröffentlicht und diese zuletzt im Jahr 2021 zu der heute gültigen Version aktualisiert [SAE, J3016 APR2021]. Teil der Norm ist auch eine Klassifizierung von automatisierten Straßenfahrzeugen. Die Klassifizierung beschreibt sechs Stufen, welche in Tabelle 1 dargestellt sind, und die Mindestanforderungen an ein Fahrzeug für die Einordnung in diese.

Was bei der Betrachtung der SAE-Stufen auffällt: Bei der Benennung der einzelnen Stufen wurde gänzlich auf den Begriff der Autonomie verzichtet. Stattdessen wird auf den beiden höchsten Stufen, auf welchen ein Fahrzeug, nach der oben genannten Definition der Begrifflichkeit der Autonomie, autonom entscheidet und handelt, von *hoher* und *vollständiger Fahrautomatisierung* gesprochen. Auch

Tabelle 1: Automatisierungsgrade von Straßenfahrzeugen nach [SAE, J3016 APR2021].

Bereich	Stufe	Bezeichnung	Beschreibung
Manuelles Fahren	0	Keine Fahrautomatisierung	Das Fahrzeug wird vollständig durch den Fahrer gesteuert.
Unterstützung	1	Fahrer-Assistenzsysteme	Übernahme von der Teilaufgabe der Quer- **oder** der Längsbewegungskontrolle durch ein Fahrautomatisierungssystem unter bestimmten Bedingungen. Der Fahrer muss die verbleibenden Fahraufgaben ausführen und im Zweifelsfall die Aufgabe übernehmen.
	2	Teil-Fahrautomatisierung	Übernahme von den Teilaufgaben der Quer- **und** der Längsbewegungskontrolle durch ein Fahrautomatisierungssystem unter bestimmten Bedingungen. Der Fahrer muss weiterhin die Umgebung beobachten, auf diese reagieren und im Zweifelsfall die Aufgaben übernehmen.
Automatisierung	3	Bedingte Fahrautomatisierung	Übernahme der gesamten Fahrzeugsteuerung durch ein Fahrautomatisierungssystem unter bestimmten Bedingungen. Der Fahrer **muss** weiterhin die Aufgaben übernehmen, wenn ihn das System dazu auffordert oder ein Fehler auftritt.
	4	Hohe Fahrautomatisierung	Übernahme der gesamten Fahrzeugsteuerung durch ein Fahrautomatisierungssystem unter bestimmten Bedingungen. Der Fahrer **muss nicht** für die Übernahme der Aufgaben zur Verfügung stehen.
	5	Vollständige Fahrautomatisierung	Übernahme der gesamten Fahrzeugsteuerung durch ein Fahrautomatisierungssystem **unabhängig** von den Umgebungsbedingungen. Der Fahrer muss nicht für die Übernahme der Aufgaben zur Verfügung stehen.

[17] `https://dwds.de/wb/Autonomie` [letzter Zugriff: 17.02.26]
[18] `https://www.sae.org/` [letzter Zugriff: 17.02.26]

Tabelle 2: Automatisierungsgrade von Luftfahrzeugen nach [EASA, 2023].

Stufe		Bezeichnung	Beschreibung
1	A	Assistenz des Menschen	Steigerung der menschlichen Leistung
	B		Unterstützung des Menschen bei der Entscheidungsfindung und Handlungsauswahl
2	A	Mensch-KI-Zusammenarbeit	Kooperation von Menschen und KI-basierten Systemen
	B		Kollaboration von Menschen und KI-basierten Systemen
3	A	Fortgeschrittene Automatisierung	Das KI-basierte System trifft selbstständig Entscheidungen und führt Aktionen selbstständig aus. Diese können vom Menschen aber überschrieben werden können.
	B		Das KI-basierte System trifft selbstständig Entscheidungen und führt Aktionen selbstständig aus. Diese können nicht vom Menschen überschrieben werden.

Tabelle 3: Autonomiegrade von Schiffen nach [IMO, 2021b].

Grad	Beschreibung
1	Schiff mit automatisierten Prozessen und Entscheidungshilfen: Personal ist an Bord um Systeme und Funktionen an Bord zu bedienen und zu steuern. Einige Vorgänge können automatisiert und zeitweise unbeaufsichtigt sein, aber es müssen zu jeder Zeit qualifizierte Personen an Bord sein, die bereit sind, die Kontrolle zu übernehmen.
2	Ferngesteuertes Schiff mit Personal an Bord: Das Schiff wird von einem anderen Ort aus gesteuert. Qualifiziertes Personal ist an Bord, um die Kontrolle zu übernehmen und die Systeme und Funktionen an Bord zu bedienen.
3	Ferngesteuertes Schiff ohne Personal an Bord: Das Schiff wird von einem anderen Ort aus gesteuert und betrieben. Es befindet sich kein qualifiziertes Personal an Bord.
4	Vollständig autonomes Schiff: Das Betriebssystem des Schiffes ist in der Lage Entscheidungen zu treffen sowie Aktionen selbst zu bestimmen und durchzuführen. Es befindet sich kein Personal an Bord. Das Schiff wird während des Betriebs von einem entfernten Ort durch Menschen überwacht.

die Agentur der Europäischen Union für Flugsicherheit (engl.: European Union Aviation Safety Agency, EASA) hat mit der Aktualisierung ihrer *Artificial Intelligence Roadmap* (dt.: Fahrplan für künstliche Intelligenz) im Mai 2023 den Begriff der Autonomie aus dem darin vorgeschlagenen Klassifizierungsschema für KI-Anwendungen in der Luftfahrt gestrichen und durch den Begriff der Automatisierung ersetzt (vgl. Tabelle 2).

Die Internationale Seeschifffahrtsorganisation (engl.: International Maritime Organization, IMO) ist eine Sonderorganisation der Vereinten Nationen, deren Einsatz u. a. der Verbesserung der maritimen Sicherheit, der Vermeidung von Verschmutzungen durch die Schifffahrt sowie allen anderen nicht rein wirtschaftlichen Interessen der Handelsschifffahrt gilt. Durch sie wurden wichtige Regelwerke wie die *International Convention for the Safety of Life at Sea* (SOLAS) und die *International Regulations for Preventing Collisions at Sea* (COLREG) erstellt und herausgegeben. Die IMO arbeitet aktuell unter anderem aktiv daran, einen regulatorischen Rahmen zu schaffen, um den sicheren und effizienten Einsatz von KI und Automatisierungstechnologien auf Schiffen zu gewährleisten. Begonnen wurde dieser Prozess im Jahr 2018 damit, dass eine Regulierungsstudie für automatisierte Schiffe angefertigt wurde, um bestehende Vorschriften im Hinblick auf die Anwendung solcher Technologien zu überprüfen und festzustellen, ob Anpassungen erforderlich sind. Von den ersten Ergebnissen an bis heute wurde in veröffentlichten IMO-Dokumenten stets der Oberbegriff der maritimen autonomen Oberflächenschiffe (engl.: Maritime Autonomous Surface Ships, MASS) genutzt – welcher, trotz seiner nicht vollständig zutreffenden Nutzung des Autonomie-Begriffes, weitverbreitet genutzt wird. Aktuell definiert die IMO für automatisierte Schiffe die in Tabelle 3 dargestellten Abstufungen [IMO, 2021b].

Ergebnisse zum Thema Sicherheit des Betriebs von automatisierten Schiffen wurden von der IMO bisher nur in Form von Protokollen veröffentlicht. Um eine regulatorische Basis zu schaffen, plant die IMO allerdings die Veröffentlichung nichtobligatorischer Vorgaben für den Betrieb von automatisierten Schiffen bis Ende 2024 und die Veröffentlichung verpflichtender Vorgaben bis zum 1. Januar 2028 [IMO, 2022]. Da die Abkürzung MASS weitverbreitet ist, wird diese ebenfalls im weiteren Verlauf dieser Arbeit genutzt. Aufgrund bisher fehlender offizieller Definitionen ist der Abkürzung hier aber folgende Bedeutung zugemessen: *Maritime Automated Surface Ship* (dt.: maritimes automatisiertes Oberflächenschiff). Diese Änderung ist damit begründet, dass der Oberbegriff nicht nur für vollständig automatisierte (d. h. autonome) Schiffe genutzt wird, sondern auch für die vorhergehenden Abstufungen, welche lediglich als automatisiert anzusehen sind. Zusätzlich wird so eine Homogenisierung mit den Begrifflichkeiten anderer Verkehrsdomänen vorangetrieben.

Nicht nur zwischen den verschiedenen Verkehrsdomänen herrscht eine gewisse Diskrepanz bzgl. der Einordnung von Automatisierungen. Auch innerhalb der Domäne des maritimen Transports herrscht noch begriffliche und taxonomische Uneinigkeit in Bezug auf automatisierte maritime Fahrzeuge [Rødseth & Wennersberg, 2023; Askari & Hossain, 2022]. Dies gilt zum aktuellen Zeitpunkt auch über die bereits angerissenen disjunkten akademischen Forschungsprojekte (vgl. u. a. [Vagia et al., 2016]) hinaus, für regulierende und standardisierende Entitäten [Baugehn & Tettenborn, 2021; Wright, 2020], die Industrie und wirtschaftlich handelnde Unternehmen. So definiert *Rolls Royce* fünf Level der Automatisierung [Teo, 2017] und nutzte dabei den Begriff der Autonomie auch für Schiffe, welche nach der oben genannten Wortbedeutung lediglich automatisiert sind: (L0) No Autonomy, (L1) Partial Autonomy, (L2) Conditional Autonomy, (L3) High Autonomy, (L4) Full Autonomy. Zur Klassifikation werden dabei die Funktionsbereiche Monitoring, Reporting, Decision Support, Decision Making und Decision Execution betrachtet.

Als regulierende Entität hat die internationale Klassifikationsgesellschaft *DNV*[19] fünf Level veröffentlicht. Diese orientieren sich weniger an abstrakten Begriffen wie *partial* und *high*, sondern konzentrieren sich stattdessen auf das klar differenzierbare Zusammenspiel zwischen Mensch und Maschine: „(M) Manually operated function, (DS) System decision supported function, (DSE) System decision supported function with conditional system execution capabilities (human in the loop, required acknowledgement by human before execution), (SC) Self-controlled function (the system will execute the operation, but the human is able to override the action. Sometimes referred to as ‚human on the loop'), (A) Autonomous function (the system will execute the function, normally without the possibility for a human to intervene on the function level)." [DNV, CG-0264:2021]

Eine ähnliche Richtung schlägt die Zertifizierungsgesellschaft *Bureau Veritas* ein und definiert in ihren 2019 zuletzt aktualisierten *Guidelines for Autonomous Shipping* ebenfalls fünf Level, welche durch die Aufteilung von Verantwortlichkeiten auf Mensch und System klar abgrenzbar sind [Bureau Veritas, 2019]. Dabei wird entsprechend der Empfehlung aus § 7 des SAE-Dokuments J3016™ SEP2016 *Surface Vehicle Recommended Practice – Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles* ebenfalls der Begriff der Autonomie vermieden und stattdessen der Begriff der Automation genutzt. Dabei wird die höchste Stufe erstmals lediglich als *Full Automation* bezeichnet, statt von Autonomie zu sprechen.

[19] `https://www.dnv.com/maritime/` [letzter Zugriff: 17.02.26]

Nachdem die Klassifikationsgesellschaft und Risikomanagement-Organisation *Lloyd's Register* anfänglich im Jahr 2016 ebenfalls eine Einteilung in Autonomiestufen verwendete und veröffentlichte, ist sie mittlerweile dazu übergegangen, den möglichen digitalen Zugriff auf Daten und Steuersysteme des Schiffes in ein System mit Stufen von null bis fünf einzuteilen, um so die eigenen Prozesse besser auf Herausforderungen der Qualitätssicherung auszurichten [Sivori & Brunton, 2023; Lloyd's Register, 2019]. Das American Bureau of Shipping definiert in seinen Veröffentlichungen lediglich drei *System Autonomy Levels*: (Manual), Smart, Semi-Autonomy, Full Autonomy. Die einordnende Entscheidung wird auch hier anhand der Aufteilung der Verantwortungen für Tätigkeiten wie das Monitoring, die Analyse, die Entscheidung und die Ausführung von Aktionen auf den Menschen und die Maschine getroffen. [American Bureau of Shipping, 2020]

Einige Klassifikationsschemata ordnen dabei Automatisierung, Fernsteuerung und die Anwesenheit von Personal an Bord in eine gemeinsame Skala ein. So schlagen Rødseth et al. in [Rødseth et al., 2022] eine Skala vor, welche unter anderem die Level *Remote Supervision*, *Remote Control* und *Periodically Unattended* enthält. Diese Arbeit geht davon aus, dass die An- oder Abwesenheit von Personal an Bord des Schiffes keinen Einfluss auf den Grad der Automatisierung hat – vorstellbar wäre etwa ein Schiff, das vollkommen autonom vom Start- zum Zielhafen reist, dabei aber Personal auf dem Schiff ist, welches sich um die Ladung kümmert. Ob ein Schiff *manned* oder *unmanned* ist, wird im weiteren Verlauf daher nicht von Bedeutung sein. Ähnlich verhält es sich mit der Fernsteuerbarkeit: Ein Schiff, welches von einem entfernten Ort (z. B. einer landseitigen Kommandozentrale) gesteuert wird, wird nach wie vor von einem Menschen gesteuert und kann nicht automatisch als automatisiert oder gar autonom angesehen werden. Das heißt für diese Arbeit, dass es keinen Unterschied macht, ob der Mensch, welcher die Verantwortung für die Steuerung des Schiffes hat, sich physisch auf dem Schiff befindet oder nicht. Ferngesteuerte Schiffe stellen eine eigene Kategorie dar, welche nicht weiter explizit betrachtet wird.

Auf Basis der aufgezeigten Vorarbeiten, Erkenntnisse sowie deren Einschränkungen und der genannten Kritik an vorhandenen Skalen und Taxonomien wurde die in Abbildung 6 dargestellte Taxonomie erstellt, welche als Basis für den weiteren Verlauf der Arbeit dienen soll. Im oberen Teil der Baum-

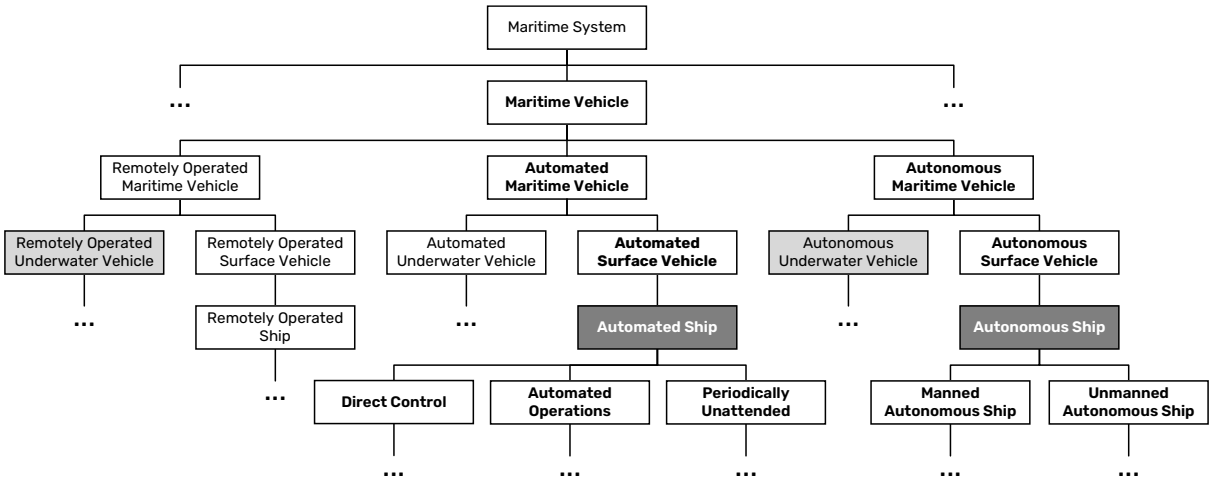


Abbildung 6: Taxonomie maritimer Fahrzeuge nach der Art und Weise, wie diese gesteuert und bedient werden. Die hellgrauen Rechtecke stellen bereits gebräuchliche Begriffe dar. Dunkelgraue Rechtecke repräsentieren die Arten maritimer Fahrzeuge, die im Fokus der Betrachtung dieser Arbeit stehen. Der abgeänderte Begriff *Maritime Automated Surface Ship* (MASS) umfasst einfachheitshalber sowohl die Kategorie *Automated Ship* als auch die Kategorie *Autonomous Ship*.

Tabelle 4: Skala des Grades der Automatisierung von Wasserfahrzeugen nach [One Sea, 2022].

Level	Beschreibung	Verantwortlichkeit (Mensch)
0	Basisbetrieb	Mensch steuert das Schiff
1	Unterstützter Betrieb	Hands-On, Eyes-On, Mind-On
2	Teilweise Automatisierung	Hands-Off at-times, Eyes-On, Mind-On
3	Zustandsabhängige Automatisierung	Hands-Off, Eyes-Off at times, Mind-On
4	Hohe Automatisierung	Hands-Off, Eyes-Off, Mind-Off at times
5	Autonomer Betrieb	Hands-Off, Eyes-Off, Mind-Off (Humans-Off)

struktur wird dabei auf den Begriff des Vehikels zurückgegriffen, analog zu den in anderen Verkehrsdomänen gebräuchlichen Begriffen wie dem des unbemannten Luftfahrzeugs (engl.: Unmanned Aerial Vehicle, UAV) und dem des unbemannten Landfahrzeugs (engl.: Unmanned Ground Vehicle, UGV).

Um den Grad der Automatisierung für ein *Automated Ship* dabei weiter genauer differenzieren zu können, wurde auf die von der One Sea Association [One Sea, 2022] vorgeschlagene Skala zurückgegriffen (vgl. Tabelle 4). Diese erfüllt alle zuvor genannten, kritisch herausgearbeiteten Aspekte und konzentriert sich gänzlich auf technische Automation und die Verlagerung von Verantwortungen vom Menschen hin zur Maschine. Die Verantwortung wird dabei aufgeteilt in die aktive Steuerung (Hands-On/-Off), die Beobachtung der Umwelt und des Schiffes (Eyes-On/-Off) und das Treffen von Entscheidungen (Mind-On/-Off). Level 0 entspricht dabei einem manuell gesteuerten Schiff ohne Automation. Dabei ist wichtig zu erwähnen, dass die Automation hier ausschließlich auf Aufgaben der Planung, Navigation und Bewegungssteuerung des Schiffes bezogen ist. Andere Teilsysteme können auch in dieser Stufe bereits automatisiert sein, wie die interne Motorsteuerung mit einer automatisierten Einspritzanlage. Level 5 entspricht dem *Autonomous Ship* aus Abbildung 6.

6.2 SCHIFFSFÜHRUNGSSYSTEME

Einigkeit scheint darüber zu herrschen, dass nicht alle Systeme eines Schiffes den gleichen Automatisierungsgrad aufweisen müssen. Ein denkbares Szenario ist z. B., dass die Steuerung des Schiffes bereits vollständig automatisiert durchgeführt wird, die Brandbekämpfung aber weiterhin von Menschen durchgeführt wird [Rødseth et al., 2022]. Weil vordergründig die vollständige Automatisierung der schiffführenden Aufgaben, also der Navigation, Führung und Kontrolle (engl.: Guidance, Navigation and Control, GNC), von Interesse ist und in Zukunft vermutlich den Großteil der simulativ zu testenden Systeme ausmachen wird, werden ebensolche Schiffführungssysteme nachfolgend genauer betrachtet.

Da vor der Einführung und bei Abwesenheit von automatisierten Systemen Menschen die Führung von Schiffen übernehmen, ist zunächst ein Blick auf den allgemeinen Prozess menschlicher Informationsverarbeitung sinnvoll. Parasuraman et al. schlagen in ihrer Arbeit zu möglichen Arten menschlicher Interaktionen mit automatisierten Systemen ein stark vereinfachtes 4-Ebenen-Modell zur Beschreibung des menschlichen Informationsverarbeitungsprozesses vor (vgl. Abbildung 7). In der ersten

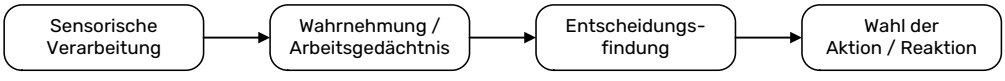


Abbildung 7: Abstraktes 4-Ebenen-Modell des menschlichen Informationsverarbeitungsprozesses nach [Parasuraman, Sheridan und Wickens, 2000] (Übers. d. Verf.)

Phase erfolgt die Wahrnehmung von Daten aus einer Vielzahl von Quellen. Diese Phase umfasst die Einordnung der Sinnesrezeptoren innerhalb der Umwelt, die sensorische Verarbeitung, die erste Vorverarbeitung der sensorischen Daten sowie die selektive Aufmerksamkeit. In der zweiten Phase erfolgt die bewusste Wahrnehmung der kontextuell angereicherten sensorischen Daten sowie die Verarbeitung dieser im Arbeitsgedächtnis[20]. Diese Phase beinhaltet zudem kognitive Operationen, die vor der Entscheidung stattfinden, wie die Erkennung von Bekanntem und Schlussfolgerungen. In der dritten Phase werden Entscheidungen auf Basis dieser Informationen getroffen. Die vierte und finale Phase umfasst die Umsetzung einer reaktiven oder proaktiven Handlung, welche mit der getroffenen Entscheidung korrespondiert und sich auf die Umwelt auswirkt. [Hasselhorn & Grube, 2003]

Diese vierstufige Modellierung der menschlichen Informationsverarbeitung stellt bewusst eine stark vereinfachte Darstellung der tatsächlich von Kognitionspsychologen identifizierten komplexen Zusammenhänge dar [Hasselhorn & Grube, 2003]. Obwohl sie daher nicht für die kognitionswissenschaftliche Anwendung geeignet ist, kann sie helfen, menschliches Verhalten bei der Bewältigung von Aufgaben, welche traditionell durch Menschen erledigt werden, wie das Steuern eines Schiffes, abstrahiert zu verstehen, zu modellieren und zu automatisieren. Zu diesem Zweck kann die übergreifende Fahraufgabe in die Teilaufgaben *Navigation, Führung und Kontrolle* (GNC) unterteilt werden. Identifiziert man anschließend für jede der drei Teilaufgaben, welche Daten und Verarbeitungsschritte für jede der vier Ebenen des Informationsverarbeitungsprozesses nötig sind, so ergibt sich die Grundlage für die Architektur des Systems, welches die Fahraufgabe unterstützen oder übernehmen soll.

Um Daten aus der Umwelt zu gewinnen, werden Sensoren benötigt, welche die Aufgaben der Sinnesorgane eines Menschen übernehmen. Nachdem eine Entscheidung getroffen und eine auszuführende Aktion/Reaktion ausgewählt wurde, benötigt das System zudem Aktoren, mit welchen die Bewegung des Schiffes in der realen Welt beeinflusst werden kann. Ein solches System ist daher nach der Definition von Kopetz [2019] ein *Cyber-Physisches System* (engl.: Cyber-Physical Systems, CPS). Ein automatisiertes Schiffsführungssystem im Sinne eines CPS besteht aus Teilsystemen und Komponenten wie Sensoren, Aktoren, Prozessoren, elektronischen Schaltungen, integrierten Schaltkreisen, Software und Kommunikationsinfrastruktur. Die Integration dieser elektronischen, softwarebasierten und physikalischen Komponenten führt zu einem gemeinsamen Systemverhalten [Brinkmann, 2018]. Die Interaktion und Kommunikation zwischen den Systemkomponenten, dem Schiff und der Umwelt ist in Abbildung 8, ausgehend von einer Unterteilung in die GNC-Teilaspekte, dargestellt.

Navigation: Die oberste Ebene ist für die strategische Routenplanung und -überwachung zuständig und umfasst die Ermittlung und Anpassung möglicher Routen. Bei diesem Prozess werden das Wasserstraßennetz und verschiedene weitere Parameter, wie Wettervorhersagen oder der geschätzte Treibstoffverbrauch, berücksichtigt, um möglichst den optimalen Weg von einem Ausgangspunkt zu seinem Zielpunkt zu finden. Die Navigationskomponente stützt sich auf detaillierte Informationen über das Wasserstraßennetz, einschließlich Seewege, statische Hindernisse und Tiefenkonturen aus Seekarten. Die berechnete Route wird zusammen mit ihren zeitlichen Aspekten an die nachfolgende Ebene übergeben. Externe Faktoren wie veränderte Wetterbedingungen oder außerordentliche behördliche Auflagen können nachträgliche Änderungen an der geplanten Route erforderlich machen.

[20] Eine einheitliche Definition einiger Aspekte des Arbeitsgedächtnis eines Menschen existiert noch nicht. Allen gegenwärtig diskutierten Arbeitsgedächtnismodellen ist aber gemeinsam, dass sie unter Arbeitsgedächtnis ein internes kognitives System verstehen, welches ermöglicht mehrere Informationen vorübergehend zu speichern und miteinander in Beziehung zu setzen. [Hasselhorn & Grube, 2003].

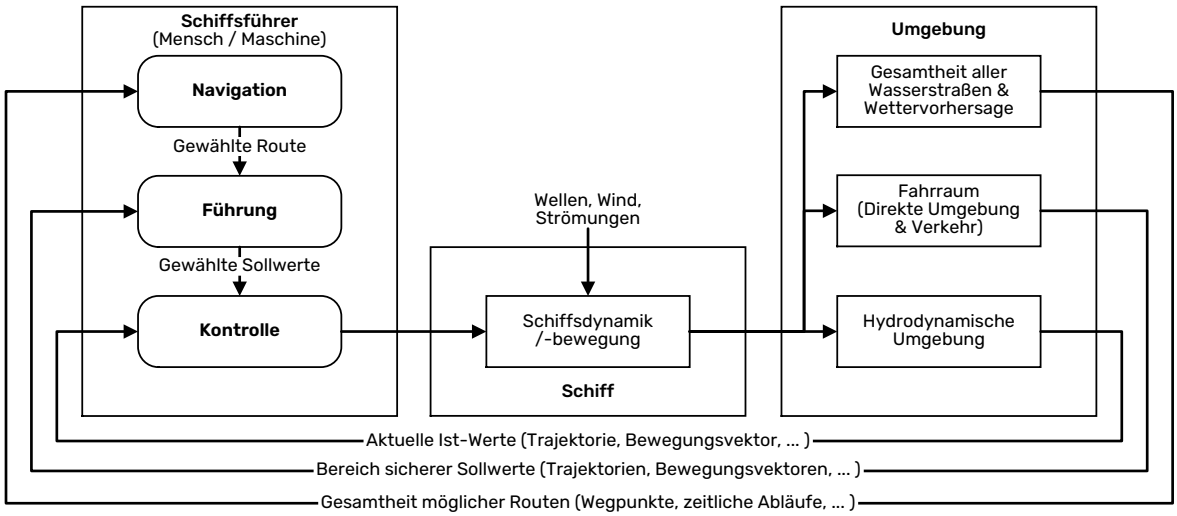


Abbildung 8: Drei-Ebenen-Hierarchie für Fahraufgaben nach [Donges, 2015] angepasst auf die Konzepte und Eigenschaften maritimen Verkehrs (angelehnt an [Brinkmann, 2018] mit zusätzlicher strengerer Integration der Teilaufgaben eines üblichen GNC-Konzepts, wie es u. a. in [Fossen, 2011] beschrieben ist). Zu erkennen sind die Feedbackschleifen aus Wahrnehmung der Umgebung und Ausführung von Aktionen, welche sich auf die Umwelt auswirken.

Führung: Auf der mittleren Ebene des Drei-Schichten-Modells arbeitet die Führungskomponente. Sie berücksichtigt den verfügbaren Navigationsraum und die Verkehrsbedingungen, die durch Faktoren wie dynamische Hindernisse beeinflusst werden. Das System überwacht alle Abweichungen von der strategisch gewählten Route und berechnet kontinuierlich die gewünschte Position, Geschwindigkeit und Beschleunigung des Schiffes. Sensoren an Bord des Schiffes erfassen Parameter wie Beschleunigung und Rotationsgeschwindigkeit, um ein möglichst genaues Bild der aktuellen Lage des Schiffes selbst aggregierend zu erzeugen. Das Situationsbewusstsein bezüglich der Umwelt des Schiffes kann u. a. durch den Einsatz von Radar-, Lidar- und Kamerasystemen erreicht werden. Weitere Sensoren werden zur Messung von Umweltfaktoren wie Windrichtung und -geschwindigkeit eingesetzt. Die Führungskomponente ist für die Berechnung sicherer Trajektorien und Bewegungsvektoren zuständig, die dann an die Kontrollsysteme weitergeleitet werden.

Kontrolle: Die unterste Ebene befasst sich mit unmittelbaren Reaktionen auf Differenzen zwischen dem aktuellen Ist-Zustand des Schiffes und dem vorgegebenen Soll-Zustand. Die hydrodynamische Umgebung wirkt u. a. durch Wellen und Strömungen als externe Kraft auf das Schiff ein, wodurch dessen Position beeinflusst werden kann. Das stabilisierende Ausgleichen der Einflüsse externer Kräfte durch weiteres Anpassen der durch die eigenen Aktoren erzeugten Kräfte ist auf dieser Ebene besonders wichtig. Ebenso beeinflussen auch die eigene Schiffsdynamik, die unmittelbare Umgebung und das Wetter die direkte Bewegungssteuerung des Schiffes. Diese Stabilisierung erfolgt auf Basis kontinuierlicher Überwachung der eigenen Schiffsbewegungen durch Auswertung von Sensordaten. Die Führungskomponente liefert Soll-Werte in Form von Trajektorien und Bewegungsvektoren als Grundlage für die Berechnung erforderlicher Maßnahmen. Diese zur Erreichung der Soll-Werte nötigen Maßnahmen werden wiederum in direkte Steuerbefehle für die Aktuatoren des Schiffes übersetzt. Die Ausführung der Befehle durch die Aktuatoren wirkt sich schlussendlich auf die Schiffsbewegung aus. [Fossen, 2011] Auf dieser Ebene wird also sichergestellt, dass die angewandten Kräfte zur Einhaltung der vorgegebenen Soll-Werte führen sowie, dass die Stabilität und Manövrierfähigkeit des Schiffes aufrechterhalten werden.

6.3 SYSTEMBEGRIFF

Der Begriff des Systems wurde in den vorangegangenen Abschnitten bereits mehrfach genutzt, und auch außerhalb dieser Arbeit sind in Bezug auf Fahrzeuge immer wieder Begriffe wie *Fahrsystem*, *Transportsystem*, *Verkehrssystem*, *Kontrollsystem* und *Navigationssystem* zu finden. Mit einem thematischen Bezug zum Testen von Neuentwicklungen ist wiederum häufig die Rede von *Testsystem* und dem *zu testenden System*. Aber auch bei einem Blick über den thematischen Tellerrand dieser Arbeit stößt man auf Begriffe wie *Soziales System* und *Ökosystem*. Die genannten Beispiele haben auf den ersten Blick wenig gemeinsam, weshalb eine klare Definition des Begriffes *System* notwendig ist. Diese zu liefern hat sich die interdisziplinäre Wissenschaft der Systemtheorie zur Aufgabe gemacht. Aufgrund der zentralen Rollen, die Systemen unterschiedlicher Ausprägungen im Rahmen der vorliegenden Thematik zukommen, werden nachfolgend einige der gängigen systemwissenschaftlichen Definitionen, Bezug nehmend auf automatisierte Schifffahrt und softwarebasierte Assistenzsysteme, betrachtet.

Allen Systemen ist gemein, dass sie aus einer Menge von Elementen bestehen. In einem technischen System können diese Elemente z. B. technische Komponenten sein, die unterschiedliche Aufgaben übernehmen. Was ein System von einer bloßen Ansammlung integrierter Komponenten abhebt, ist eine Interaktion und Interdependenz zwischen den individuellen Bestandteilen sowie die Ausrichtung auf ein gemeinsames Ziel. Daher kann der Begriff des Systems als eine Anzahl von Elementen, als Teile eines Systems, die zusammenarbeiten und interagieren, um ein bestimmtes Ziel zu erreichen, beschrieben werden [Patzak, 1982; Forrester, 1972]. Dabei ist es wichtig, zu betonen, dass das Konzept eines Systems eine ganzheitliche Perspektive einnimmt und die funktionalen Verbindungen zwischen einzelnen Elementen und dem System als Ganzes in den Fokus stellt. Diese ganzheitliche Perspektive soll u. a. deutlich machen, dass ein System die reine Summe seiner Teile funktional übertrifft:

> *„[…] this concept of […] emphasizes the functional relationships between parts and wholes, it supposes that wholes cannot be reduced to the sum of their parts or, alternatively, that a system is more than the mere sum of its parts.“*
>
> [Barceló, 2010a]

Forrester nennt dazu einige verständliche Beispiele aus dem Alltag: Ein Kraftfahrzeug ist ein System von Komponenten, die zur Durchführung von Transporten zusammenwirken. Ein Autopilot und ein Flugzeug bilden zusammen ein System, das gemeinsam auf die Flughöhe hinwirkt. Ein Lagerhaus und eine Laderampe bilden gemeinsam ein System zum Verladen von Waren auf Lastkraftwagen. [Forrester, 1972] Bei den genannten Beispielen ist klar zu erkennen, was Teil des Systems ist und was nicht. Allerdings gibt es auch Systeme, bei denen diese Abgrenzung als externer Betrachter nicht trivial ist: Betrachtet man das Antiblockiersystem eines Kraftfahrzeugs, stellt sich schnell die Frage, wo dieses beginnt und wo es aufhört. Sind etwa die Bremsbeläge selbst noch Teil des Antiblockiersystems oder werden diese lediglich indirekt durch das Antiblockiersystem beeinflusst, sind aber selbst Teil des Bremssystems, bestehend aus Hydraulikpumpe, Schläuchen, Bremszylinder etc.? Eine eindeutige Systemgrenze stellt also einen essenziellen Bestandteil einer Systembeschreibung dar. Die Systemgrenze dabei ist eine gedankliche Abgrenzung realweltlicher Gegebenheiten, mit deren Hilfe festgelegt wird, was als Systeminneres anzusehen ist. Die Systemgrenze ist dabei eine eindeutige Trennlinie zwischen dem System selbst und seiner Umgebung (vgl. Abbildung 9). [Dierßen, 2002]

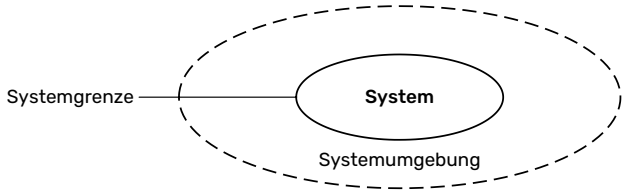


Abbildung 9: Abgrenzung eines Systems zur (für das System relevanten) Umwelt

Ist eine eindeutige Systemgrenze identifiziert, stellt sich als Nächstes die Frage, wie das Systeminnere gestaltet ist. Zuvor wurde ein System bereits als eine Menge von interagierenden Elementen beschrieben. Ein Element ist etwas, das sich durch ein Substantiv oder eine Substantivgruppe benennen lässt und über dessen Existenz sich alle informierten Stakeholder einig sind. Jedes Element als Teil eines Systems im Sinne der Systemtheorie besitzt eine Menge diskreter Eigenschaften [Patzak, 1982], auch Attribute [Koreimann, 2000] genannt, und dazugehörige konkrete Ausprägungen. Zusätzlich muss jedes Element die Fähigkeit besitzen, seinen Zustand, d. h. die Ausprägungen seiner Attribute, selbstständig proaktiv und/oder reaktiv zu verändern. Es muss also ein Verhalten besitzen. Eine Beziehung zwischen zwei Elementen ist dann gegeben, wenn das Verhalten eines Elements durch das eines anderen beeinflusst wird. Die Beziehungen zwischen Elementen können verschiedener Art sein, stellen im Kern aber immer einen Fluss von Material, Energie, Informationen oder Daten dar. [Flood & Carson, 1988] Dabei gibt es innerhalb des Systems keine Elemente, die nicht jeweils in Beziehung mit mindestens einem anderen Element stehen: Alle Elemente eines Systems wirken aufeinander, miteinander, nebeneinander, nacheinander oder ggf. auch gegeneinander [Reich, 2010] (vgl. Abbildung 10).

In anwendungsorientierten technischen Fachgebieten werden die Elemente oft als Komponenten [Harvey & Stanton, 2014; Maier, 1998; Patzak, 1982], im Sinne der Definition einer Komponente als *bestimmter Bestandteil eines Ganzen*[21], oder als Objekte [Dierßen, 2002; Gipser, 1999] bezeichnet. Im weiteren Verlauf dieser Arbeit wird der Begriff der Komponente genutzt, da das Konzept einer Komponente für Personen mit (software-)technischem Hintergrund am verständlichsten sein sollte.

Zusätzlich zu den Wirkungen zwischen den Komponenten, welche die Beziehungen der Komponenten untereinander ausmachen, wird das System aus der Systemumgebung heraus beeinflusst. Wirken diese Einflüsse auf den ersten Blick auf das System als Ganzes, so lassen sich doch in der Regel ein oder mehrere Komponenten bestimmen, welchen diese zugeordnet werden können. Diese Wirkungen werden als *Eingangsgrößen* des Systems bezeichnet (vgl. Abbildung 10). Eingangsgrößen können weiter anhand des Grades der Kontrolle, die das System selbst oder externe Nutzer über sie haben, spezifi-

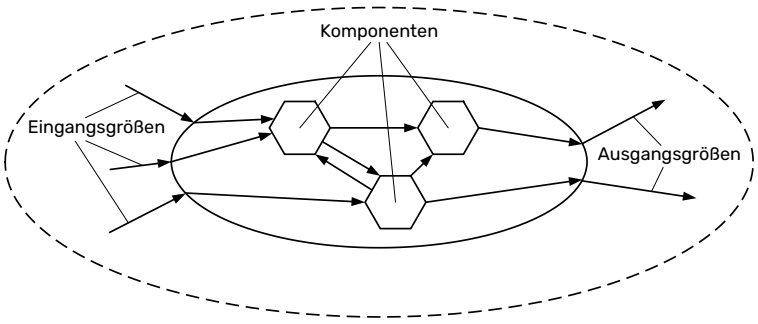


Abbildung 10: Komponenten eines Systems (angelehnt an [Dierßen, 2002])

[21] „Komponente – Schreibung, Definition, Bedeutung, Etymologie, Synonyme, Beispiele | DWDS“ `https://www.dwds.de/wb/Komponente` [letzter Zugriff: 17.02.26]

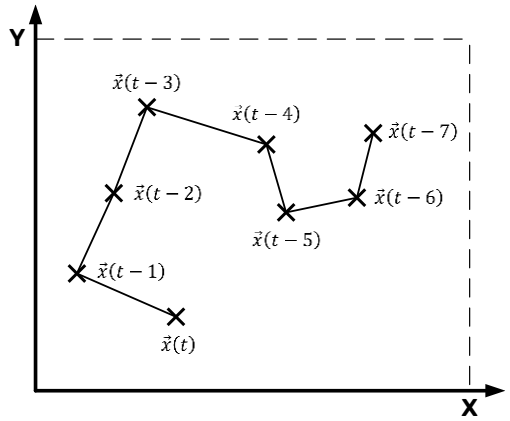
ziert werden: Können die Eingangsgrößen im Prinzip beliebig vorgegeben werden, werden sie als *Steuergrößen* bezeichnet [Dierßen, 2002]. Solche, die nicht kontrollierbar sind, werden hingegen als *Störgrößen* bezeichnet. Im Anschluss an die Verarbeitung der Eingangsgrößen wirkt ein technisches System in aller Regel zudem auch auf seine Umwelt ein. Diese beeinflussenden Wirkungen auf die Systemumgebung werden als Ausgangsgrößen des Systems bezeichnet [Gipser, 1999] (vgl. Abbildung 10). Die unterschiedlichen Größen können bestimmte Werte annehmen, wodurch sie quantifizierbar sind. Die Größen werden dabei durch Angabe eines Zahlenwertes und der zugehörigen Einheit definiert. Bei dynamischen Systemen sind die beteiligten Größen zeitabhängig, verändern sich also mit dem Fortschreiten der Zeit. Diese werden dann auch als Signale bezeichnet. Der zeitliche Verlauf aller Signale eines dynamischen Systems kann als dynamischer Prozess bezeichnet werden. [Gipser, 1999]

Vorläufig zusammengefasst besteht die Beschreibung eines Systems, wie es hier definiert und im weiteren Verlauf genutzt wird, also aus den folgenden Teilen:

- Eine klare Abgrenzung des Systems zu seiner Umwelt (Systemgrenze)
- Eine Menge an Eingangsgrößen des Systems (äußere Kopplung)
- Eine Menge an Ausgangsgrößen des Systems (äußere Kopplung)
- Eine Menge an Komponenten mit einer Menge an Attributen und einem Verhalten
- Einer Menge an Beziehungen zwischen den Komponenten (innere Kopplung)

Die Einteilung in Komponenten und die Definition ihrer Attribute und ihres Verhaltens sowie die genaue Definition der Eingänge und Ausgänge sowie der inneren und äußeren Kopplungen sind eine der Aufgaben der Modellbildung [Gipser, 1999]. Modelle und Modellbildung werden im weiteren Verlauf u. a. in den Abschnitten 7.3.1 und 7.4 erneut thematisiert.

Attribute von Komponenten, die gemeinsam den aktuellen Zustand des Systems abbilden, werden auch die *Zustandsvariablen* des Systems $x_1, \ldots, x_n$ genannt (z. B. die aktuelle Drehzahl des Motors eines Fahrzeugs, der freie Speicherplatz auf einer Festplatte innerhalb eines Servers, die aktuell von einer Sensorkomponente gemessene Flughöhe eines Flugzeugs). Die Menge der aktuellen Werteausprägung $x_i(t)$ jedes der Attribute dieser Menge zu einem Zeitpunkt t kann somit als ein *Zustandsvektor* $\vec{x}(t)$ dargestellt werden und beschreibt den *Zustand* des Gesamtsystems zu diesem beliebigen Zeitpunkt t.



$$Zustandsvektor_t = \vec{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \end{bmatrix} = \begin{bmatrix} \mathrm{X}_t \\ \mathrm{Y}_t \end{bmatrix}$$

Abbildung 11: Minimalbeispiel einer Trajektorie in einem Zustandsraum: Der Zustand eines angenommenen Systems (z. B. ein Fahrzeug) definiert sich hier nur über dessen Position in seiner als planar angenommenen Umwelt. Die Position setzt sich entsprechend zusammen aus den X- und Y-Koordinaten in dieser Umwelt. Die Umwelt hat hier eine definierte Größe und die sich dadurch ergebenden Grenzen können von dem System nicht unter- oder überschritten werden. Der sich so ergebene Zustandsraum wird durch die X-Achse, die Y-Achse sowie die gestrichelten Linien dargestellt.

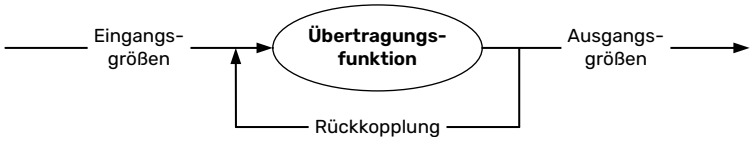


Abbildung 12: Ein offenes System als Blackbox

Die Veränderungen dieser Zustände, genauer der Werteausprägungen der Attribute, im Laufe der Zeit bilden die *Zustandstrajektorie* $X\colon [0, t_m] \to \mathbb{R}^n$. Der n-dimensionale Raum, in dem sich die Trajektorie bewegen kann, wird dabei als der *Zustandsraum* des Systems bezeichnet. [Flood & Carson, 1988]

Ein Minimalbeispiel für einen Zustandsraum ist in Abbildung 11 dargestellt. Zustandsräume zu testender Systeme und die mit diesen verbundenen Herausforderungen sind in Abschnitt 7.2.2 das zentrale Thema. Gegenstand der Untersuchung von Systemen ist dabei meistens die Übertragung von Eingangssignalen in Ausgangssignale, also die Reaktion eines Systems auf den aktuellen Zustand seiner Umwelt. Dabei ist es häufig nicht ausreichend, das System von seiner üblichen Umgebung und dessen zeitlichem Verlauf entkoppelt zu betrachten, indem z. B. im Rahmen eines Tests ein einzelner Satz vordefinierter Eingangssignale an das System übergeben wird und die erzeugten Ausgangssignale einmalig mit den bekannten, zu erwartenden Ausgangssignalen verglichen werden. Dies ist unter anderem darin begründet, dass offene Systeme in der Regel zusätzlich einer Rückkopplung unterliegen (vgl. Abbildung 12). Offene Systeme tauschen Material, Information oder Energie über ihre Systemgrenze hinweg mit ihrer Umwelt aus [Flood & Carson, 1988]. Das heißt, dass ihr Einwirken auf ihre Umwelt dazu führen kann, dass nachfolgende Eingangssignale anders ausgeprägt sind, als sie es ohne die Beeinflussung des betrachteten Systems wären. Im Gegensatz dazu stehen geschlossene Systeme, welche hauptsächlich in rein theoretischen Betrachtungen zu finden sind und für die die Systemgrenze absolut ist, also keine Beziehungen zur Umwelt zulässt [Flood & Carson, 1988].

Besonders offensichtlich ist eine solche Rückkopplung bei offenen cyberphysischen Systemen, welche durch Aktoren ihre physische Umwelt beeinflussen und verändern. So könnte z. B. ein Schiff durch seine Bewegung im Wasser eine Welle verursachen, welche von einer Kaimauer teilweise reflektiert wird und somit wieder Kräfte auf das Schiff selbst ausübt. Eines der bekanntesten durch Rückkopplung verursachten Phänomene ist das laute, hochfrequente Geräusch, welches erzeugt wird, wenn ein Mikrofon in die Nähe der Lautsprecher gebracht wird, welche das Signal eben jenes Mikrofons verstärkt wiedergeben. Das Mikrofon erzeugt Ausgangssignale, welche verstärkt durch die Lautsprecher wiedergegeben werden. Diese Veränderung der Umwelt in Form von Schallwellen erreicht wiederum als Eingangssignal das Mikrofon, wodurch sie abermals verstärkt wiedergegeben wird [Kidd & Stepan, 2014]. Aber auch rein elektronische oder softwarebasierte Systeme können Rückkopplung erfahren, vor allem dann, wenn sie Teil größerer, komplexerer Systemverbünde sind, in denen die anderen Systeme ihr Verhalten auf Basis der vom betrachteten System erzeugten Ausgangssignale ändern und somit ggf. andere Ausgangssignale erzeugen, welche wiederum als Eingangssignale für das betrachtete System dienen könnten.

Nicht immer sind nur die Beziehungen eines monolithischen Systems zu seiner Umwelt von Interesse: Mit den Innovationen der letzten Jahrzehnte wurden moderne, softwarebasierte Systeme zunehmend komplexer. Dies trifft auch auf Verkehrssysteme zu, und in diesem Kontext nicht mehr nur auf eine Kombination von eigenständigen Systemen, wie z. B. dem Gesamtsystem *Schienenverkehr* als Kombi-

nation aus der Steuerung der Lichtsignale und Schranken, den Zügen etc. [Uzuka, 2023], oder eben einem maritimen Transportsystem in seiner Gesamtheit [Mansouri et al., 2009], sondern auch auf die Fahrzeuge selbst. Diese bestehen in den meisten Verkehrsdomänen mittlerweile aus einer Kombination von komplexen mechanischen und softwarebasierten Teilsystemen, welche untereinander durch verschiedenste Kommunikationsnetzwerke verbunden sind. Bereits die ISO-Norm 17894 weist mit Blick auf maritimen Verkehr darauf hin, dass ein einzelnes modernes Schiff mehr als ein monolithisches System ist: *„The overall effect of the use of PES [programmable electronic systems] is that the ship becomes one total system of inter-linked PES and crew which work together to fulfill the operator's business goals."* [ISO, 17894:2005] Dies gilt selbst dann, wenn die Besatzung nicht einbezogen wird.

Vor allem mit Blick auf zukünftige autonome Fahrzeugsysteme, welche große Datenmengen durch Sensoren generieren, softwarebasiert verarbeiten, daraus Steuerbefehle ableiten und diese anschließend mechanisch umsetzen müssen, scheint dieser Aufwärtstrend der Komplexität und Vernetzung seinen Scheitelpunkt noch nicht erreicht zu haben. Den hohen Grad an Systemkomplexität ökonomisch vertretbar zu beherrschen, stellt die Systementwickler vor neue Herausforderungen bezüglich des Entwurfes, der Umsetzung, des Betriebs und der Wartung dieser Systeme [Rüssmeier et al., 2019; Nielsen et al., 2015; Lee, 2008]. Um Entwicklungszeiten möglichst kurz zu halten, mit dem Ziel, mit dem Markt Schritt halten zu können, setzen viele Unternehmen daher auf eine Parallelisierung der Entwicklung und Umsetzung der einzelnen Teilsysteme. Das heißt konkret, dass die Verantwortlichkeiten für die Systembestandteile im Unternehmen selbst auf eine Vielzahl von disjunkten Teams aufgeteilt oder in Gänze an externe Zulieferer ausgelagert werden (vgl. Argumentation bzgl. des Wandels von Werften von Produzenten hin zu Systemintegratoren in Abschnitt 2). Aufgrund der Tatsache, dass große Transportschiffe in der Regel Einzelanfertigungen sind und die individuellen Komponenten aufgrund der Entfernung zueinander durch das schiffsumspannende elektronische Netzwerk miteinander verbunden sind, ergibt es sich zumeist, dass diese Komponenten selbst als eigenständige Systeme umgesetzt werden. So können diese nach einmaliger Entwicklung gänzlich ohne oder mit lediglich geringen Anpassungen auf einer Vielzahl von Schiffen integriert werden. In Abbildung 13 wird ein solches modernes Containerschiff stark vereinfacht in das zuvor grafisch eingeführte Konzept eines Systems und seiner Bestandteile sowie der Systemumwelt und den Beziehungen zu dieser eingeordnet. Zu erkennen ist ebenfalls, dass einige Systemkomponenten selbst Systeme sind.

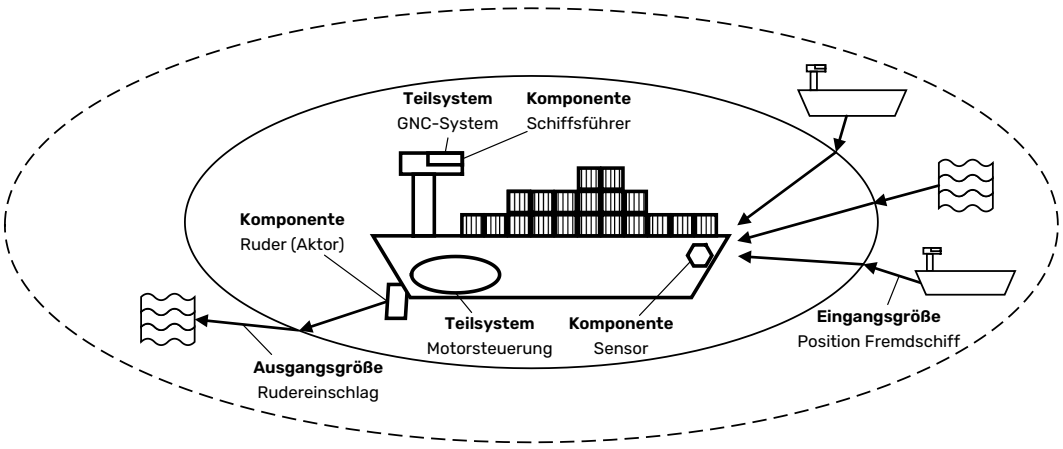


Abbildung 13: Stark vereinfachte Veranschaulichung eines Containerschiffes als System-of-Systems

Für Systeme, die als Komponenten wiederum Systeme umfassen, hat sich in der englischsprachigen Literatur der Begriff des *System-of-Systems* (SoS) etabliert [Dahmann, 2015a; ISO/IEC/IEEE, 15288:2015; Harvey & Stanton, 2014; Maier, 1998]. Die erforderlichen Eigenschaften, die ein System aufweisen muss, unterscheiden sich teilweise in den existierenden Definitionsvorschlägen. Die ISO/IEC/IEEE-Norm 15288 beschreibt ein SoS als ein System, das eine Reihe von Systemen für die Erfüllung einer Aufgabe zusammenbringt, die keines der Systeme allein vollumfänglich erfüllen kann. Jedes der beteiligten Systeme hat seine eigenen Ziele und Ressourcen sowie eine individuelle, unabhängige Verwaltung, während es sich innerhalb des SoS kooperativ einbringt und anpasst, um die übergreifenden Ziele des SoS zu erreichen. Als die wichtigsten Charakteristiken werden dabei die verwaltungstechnische und die operationelle Unabhängigkeit der teilnehmenden Systeme sowie das emergente Verhalten des SoS hervorgehoben.

Zurückblickend auf die Beschreibung des Systems *Automatisiertes Schiff* kann festgestellt werden, dass es die Anforderungen an ein SoS erfüllt. Dittmann beschreibt in einem Beitrag aus dem Jahr 2022, dass die individuellen Systeme eines solchen Schiffes (1) jeweils einem eigenen bestimmten Zweck dienen, (2) durch dedizierte Hard- und Software realisiert werden, (3) von unterschiedlichen Zulieferern entwickelt und verkauft werden sowie (4) der Informationsaustausch zwischen den Systemen durch Nachrichtenübermittlungen stattfindet [Dittmann, 2022]. Somit sind die definierenden Charakteristiken eines SoS: Die teilnehmenden Systeme haben eigene Ziele (1); die teilnehmenden Systeme haben eigene Ressourcen (2); die teilnehmenden Systeme sind operationell voneinander unabhängig (3); die teilnehmenden Systeme sind verwaltungstechnisch voneinander unabhängig (4). Ein modernes Schiff sollte daher während seines gesamten Lebenszyklus als SoS betrachtet werden. Diese Betrachtungsweise hat sich bisher nicht allgemein durchgesetzt, wurde aber sowohl für Fahrzeuge anderer Verkehrsdomänen wie moderne Flugzeuge [Barker, 2018] als auch für moderne automatisierte Schiffe bereits vorgeschlagen und angewandt [Dittmann, 2022; Hake et al., 2021; Kooij & Hekkenberg, 2021; Wennersberg et al., 2020; OFFIS e. V., 2018; Brinkmann et al., 2017; ISO, 17894:2005].

Für die Unterteilung eines Fahrzeugs als SoS und die Benennung der Bestandteile werden im weiteren Verlauf die in Abbildung 14 gezeigte Kompositionsstruktur und Begriffe in Anlehnung an [Steimle et al., 2021] verwendet.

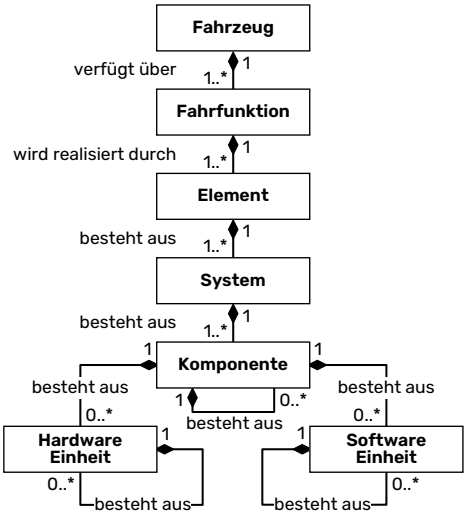


Abbildung 14: Zusammensetzung und Beziehungen automatisierter Fahrzeuge, ihrer Funktionen und Bestandteile (angelehnt an [Steimle, Menzel und Maurer, 2021])

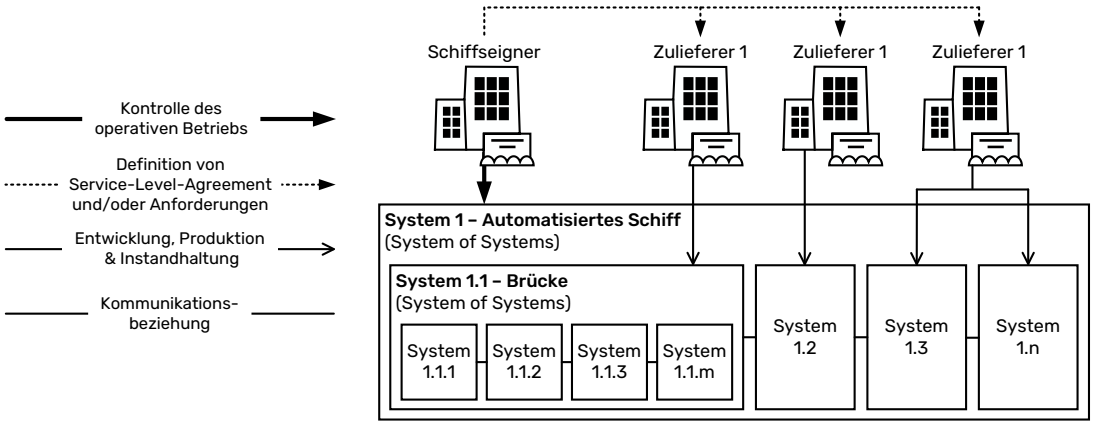


Abbildung 15: Beziehungen der an der Konstruktion und dem Betrieb eines automatisierten Schiffs beteiligten Unternehmen untereinander und zu den (Teil-)Systemen (nach [Elhadeedy und Daily, 2024])

Elhadeedy & Daily [2024] beschreiben autonome Fahrzeuge ebenfalls als SoS und begründen dies u. a. auch mit der Aufteilung der Verantwortlichkeiten für den Betrieb des Fahrzeugs sowie der Entwicklung, Instandhaltung und Integration des Fahrzeugs und seiner Teilsysteme. Sie verwenden den Begriff des *directed SoS* (dt. gezielt, gerichtet), da der Eigentümer des Endprodukts (d. h. des Fahrzeugs) die Entwicklung oder Beschaffung der konstituierenden Systeme und der anschließenden Integration dieser auf einen definierten Zweck hin ausgerichtet übergreifend leitet. Der Eigentümer des Fahrzeugs definiert die Anforderungen für die Teilsysteme, welche anschließend von Zulieferern umgesetzt oder bereitgestellt werden (vgl. Abbildung 15). Üblicherweise sind diese Zulieferer auch für die Wartung und Pflege der integrierten Teilsysteme während des Lebenszyklus des Schiffes verantwortlich – so werden ggf. Software-Updates durchgeführt, um Fehler zu beheben oder Funktionalitäten zu verbessern. Die einzelnen Systeme könnten unabhängig voneinander agieren, werden aber gezielt verwaltet, um gemeinsam einen bestimmten Zweck zu erfüllen. Dies wird spätestens durch die Tatsache ersichtlich, dass viele der auf einem Schiff eingesetzten Systeme *Off-the-Shelf*-Produkte sind (ECDIS, Motorsteuerung etc.), die lediglich beschafft und integriert werden. Das Einbinden von solchen fertigen Systemen führt dazu, dass es bzgl. Schnittstellen, Datenaustausch, Kommunikation, Timings etc. zu Diskrepanzen kommen kann, was die generelle Unabhängigkeit der einzelnen Systeme nochmals verdeutlicht. Systeme wie Electronic Chart Display and Information Systems (ECDIS) können zudem nicht nur auf Schiffen, sondern z. B. auch stationär eingesetzt werden und werden häufig nicht exklusiv für eine spezifische Systemumgebung entwickelt.

Auf dem Weg zu autonomen Schiffen werden in den nächsten Jahren zunächst die verschiedenen Stufen der Automatisierung (vgl. Abschnitt 16.1) durchlaufen werden müssen. Das heißt, dass in diesen Fällen, wie es bereits mit Blick auf [ISO, 17894:2005] angesprochen wurde, weiterhin das nötige menschliche Personal an Bord ein Teil des Systems sein wird. Ihr individuelles wie auch gemeinschaftlich organisatorisches Verhalten ist in diesen Fällen fester Bestandteil des Systems und für die Aufgabenerfüllung ebenso wichtig wie die elektronischen und mechanischen Komponenten [Henshaw, 2015]. Somit stellt ein automatisiertes Schiff ein soziotechnisches System dar. Ein soziotechnisches System ist ein Handlungs- oder Arbeitssystem, in welchem menschliche und technische Komponenten eine integrale Einheit eingehen und auf das gemeinsame Systemziel hinarbeiten [Ropohl, 2009]. Das vervollständigte Bild eines Systems, seiner Umgebung, seiner möglichen Bestandteile und der möglichen Beziehungen zwischen den beiden letztgenannten ist beispielhaft in Abbildung 16 dargestellt.

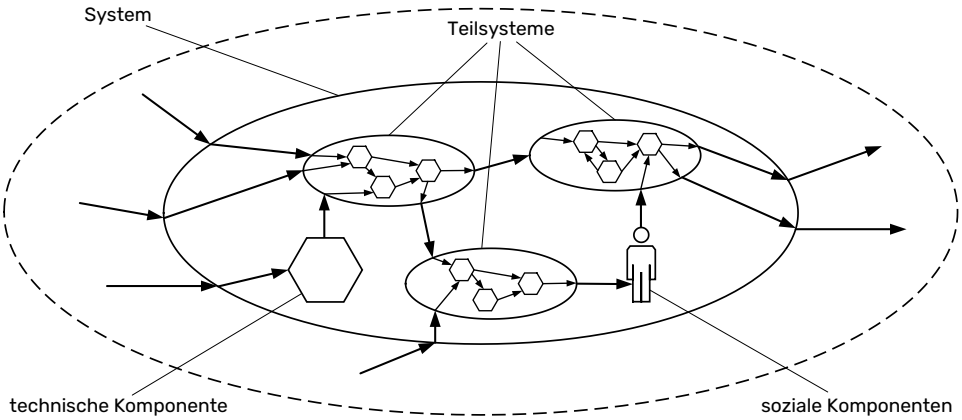


Abbildung 16: Mögliche Bestandteile eines soziotechnischen System-of-Systems

Obgleich der Fokus des Artikels [Harvey & Stanton, 2014] auf deutlich größeren soziotechnischen Systemen liegt, identifizieren Harvey & Stanton hier jedoch Herausforderungen bezüglich der Sicherstellung von Funktionalität und Sicherheit von System-of-Systems, welche sich auch auf rein technische Systeme und soziotechnische Systeme mit einer eher kleinen und/oder untergeordneten sozialen Komponente übertragen lassen:

Struktur des Netzwerks: Aus der Interaktion der Teilsysteme können sich mögliche Inkonsistenzen ergeben. Die Herausforderung besteht darin, das System-of-Systems als integriertes Ganzes zu betrachten, um die Auswirkungen auf der obersten Ebene zu überprüfen, gleichzeitig aber auch den funktional korrekten Betrieb der einzelnen Komponente sicherzustellen.

Komplexität und externe Einflüsse: Durch die Zusammenarbeit bereits inhärent komplexer Systeme steigt die Gesamtkomplexität stark an. Emergentes Verhalten ist das Ziel, führt gleichzeitig aber auch dazu, dass die inneren Strukturen des Gesamtsystems schwer nachzuvollziehen sind. Zusätzlich beeinflusst die Umgebung das Verhalten des Gesamtsystems, was aufgrund seiner Offenheit ebenfalls zu schwer hervorsehbaren Rückkopplungen führen kann.

Emergentes Verhalten: Das System-of-Systems muss zur Sicherstellung der korrekten Funktionalität als Ganzes betrachtet und getestet werden. Die hohe Komplexität sowie die Nichtlinearität der Beziehungen zwischen Handlungen und Ergebnissen können dazu führen, dass das Verhalten des Gesamtsystems praktisch nicht exakt vorhergesagt werden kann.

Informationsaustausch über Systemgrenzen: Im Falle der Integration der beteiligten Teilsysteme in einem System-of-Systems ist die Schaffung eines gemeinsamen Verständnisses bezüglich der Übertragung und Strukturierung von Informationen über die Grenzen hinweg unabdingbar.

Verantwortung für Fehler: Aufgrund des potenziellen Austausches und der Weiterverarbeitung von Informationen durch viele Teilsysteme kann die Suche nach der ursprünglichen Fehlerquelle aufwendig sein. Gleichzeitig muss die Robustheit jedes Teilsystems bezüglich möglicherweise falscher Inputdaten hoch sein.

Veränderungen: Updates einzelner Teilsysteme können zu unbeabsichtigten und nicht vorhersehbaren Verhaltensänderungen in den kooperierenden Teilsystemen führen.

6.4 VORGEHENSMODELLE

Die Menge softwaregesteuerter Funktionen in modernen Fahrzeugen nimmt stetig zu und auch die Automatisierung von Fahrfunktionen basiert auf dem Einsatz von Software. Für die Entwicklung von Software und Software-basierten Systemen existieren verschiedene Vorgehensmodelle, deren Zweck es ist, dem Entwicklungsprozess eine geordnete Struktur zu geben, um diesen übersichtlicher und beherrschbarer zu machen. Die meisten der gängigen Modelle decken dabei nicht nur die eigentliche Entwicklung ab, sondern betrachten und ordnen den gesamten Lebenszyklus der Software. Einen Überblick über einige der gängigsten Modelle für Software-Lebenszyklen bietet [Ruparelia, 2010]. Viele der Modelle können problemlos mehrheitlich auf System-Lebenszyklen übertragen werden.

Bei der Entwicklung von automatisierten und autonomen Fahrsystemen muss berücksichtigt werden, dass sie potenziell Schäden an Menschen, Materialien oder der Umwelt verursachen können. Deshalb gibt es eine Reihe etablierter Normen, die darauf abzielen, unkalkulierbare Risiken zu vermeiden, die sich aus Fehlfunktionen des entwickelten Systems ergeben könnten. Zunächst ist dabei die Norm IEC 61508 [IEC, 61508:2010] als relevant zu betrachten. Sie definiert einen allgemeinen Ansatz für die Entwicklung funktional sicherer elektronischer Systeme. Die funktionale Sicherheit ist ein Teil der Gesamtsicherheit eines Systems mit einem Fokus darauf, den korrekten Betrieb eines Systems in Abhängigkeit von seinen Umgebungsbedingungen zu gewährleisten (vgl. Abschnitt 1). Die domänenspezifische Adaption der IEC 61508 im Automobilbereich ist die ISO 26262 [ISO, 26262-1:2018], welche ebenfalls den Fokus auf das Schlüsselkonzept der funktionalen Sicherheit legt.

Um die funktionale Sicherheit zu gewährleisten, schlägt die ISO 26262 ein Entwicklungsprozessmodell vor, das auf dem weitverbreiteten *V-Modell* basiert. Das V-Modell hat seine Ursprünge ebenfalls in der reinen Softwareentwicklung [Ruparelia, 2010], hat seinen Weg in leicht angepassten Formen über die Zeit aber ebenfalls in die Systementwicklung (engl. Systems Engineering) gefunden (vgl. [VDI, 2206; Mallozzi et al., 2019; Koopman & Wagner, 2016]). Das Modell beschreibt eine Denkstruktur, die dazu dient, die im Verlauf der Entwicklung zu bewältigenden Aufgaben in einen sachlogischen Zusammenhang und eine fest definierte sequenzielle Abfolge zu bringen (vgl. Abbildung 17). Durch das Denken entlang des V-Modells erfolgt eine unmittelbare Integration zentraler Prinzipien des Systems Engineering in das individuelle Entwicklungsvorgehen [Gräßler & Oleff, 2022]. Ein ähnliches Prozessmodell wird in einem Anhang zu ISO 17894 [ISO, 17894:2005] für die Entwicklung von elektronischen Systemen für den Seeverkehr vorgeschlagen.

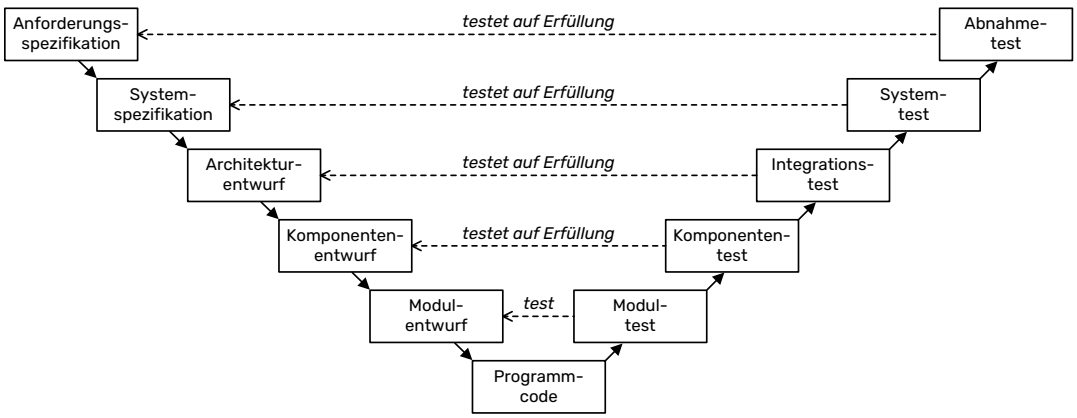


Abbildung 17: Typische Ausprägung des V-Modells, wie es u. a. in der Systementwicklung zum Einsatz kommt. Jeder Entwurfsaktivität ist eine entsprechende Testaktivität zur Überprüfung der jeweiligen Ergebnisse fest zugeordnet.

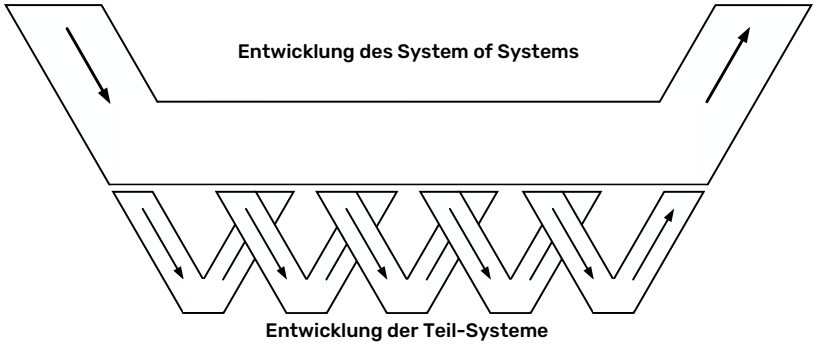


Abbildung 18: Doppel-V-Modell zur Steuerung des Entwicklungsprozesses eines System-of-Systems (in Anlehnung an [Dahmann, 2015b])

Das traditionelle V-Modell beschreibt das Vorgehen im Einklang mit der ISO-Norm für Prozesse im Lebenszyklus von Systemen in Bezug auf die Entwicklung dieser [ISO/IEC/IEEE, 15288:2015]. Das Prozess-Modell gliedert sich dabei im Allgemeinen in drei konsekutive Phasen: Zunächst wird die Entwurfsphase durchlaufen, in welcher der Systementwurf schrittweise verfeinert wird. Darauffolgend wird der entstandene Entwurf im Rahmen der Implementierungsphase im Scheitelpunkt der namensgebenden V-förmigen Anordnung der Schritte umgesetzt. Das umgesetzte System wird anschließend durch eine mehrstufige Testphase, deren Schritte zu denen aus der Entwicklungsphase isomorph sind, überprüft. Bei jedem Entwurfsschritt wird das System in kleinere Teile unterteilt. Während der Testphase wurde entsprechend umgekehrt vom Kleinen zum großen integrierten Ganzen gearbeitet. Zu Beginn jedes Entwurfsschritts wurden die Anforderungen an die zu erstellenden Artefakte ermittelt und dokumentiert. Diese wurden in der später folgenden Testphase für die entsprechenden Testaktivitäten herangezogen, um die in der Zwischenzeit erstellten Artefakte sukzessive auf die Erfüllung dieser hin zu überprüfen und somit ihre Korrektheit gegenüber den Anforderungen zu beweisen.

Im Zusammenhang mit Systems Engineering, das auf ein System-of-Systems angewandt wird, schlägt Dahmann eine Erweiterung des V-Modells zum *„Doppel-V"-Modell* vor (vgl. Abbildung 18). Dieses Modell betont die Einbettung der Entwicklungsprozesse der einzelnen teilnehmenden Systeme in den übergreifenden Entwicklungsprozess des System-of-Systems als logisch eigenständiges System im Sinne des System Engineering. Diese Darstellung kann als logische Perspektive des SoS-SE-Prozesses betrachtet werden [Dahmann, 2015b]. Diese Darstellung ist auch auf ein automatisiertes Schiff im Sinne eines directed System-of-Systems (vgl. Abschnitt 6.3) anwendbar: Im Rahmen der Planung, Konstruktion und Herstellung des Schiffes durch ein Schiffbau-Unternehmen werden zunächst Anforderungen an das gesamte Schiff definiert und entsprechende Entwürfe angefertigt. Basierend auf daraus abgeleiteten Anforderungen an die einzelnen Teilsysteme werden individuelle Entwicklungsprozesse initiiert (häufig durch zuliefernde Drittunternehmen, vgl. Abschnitt 16.3). Sind alle Teilsysteme umgesetzt und individuell getestet worden, folgt die schrittweise Integration dieser in das geplante Schiff. Die gesamte Entwicklung der Teilsysteme findet somit innerhalb der Implementierungsphase im übergreifenden Prozess des Gesamtsystems nach dem V-Modell statt.

7 BEWEIS VON FUNKTIONALITÄT UND SICHERHEIT

Viele komplexe Systeme scheitern an nicht ausreichend geprüfter Software, weshalb es möglich ist, die Relevanz von umfassender Überprüfung korrekter Funktionalität softwarebasierter Systeme anhand von historischen Beispielen für schwerwiegende Fehler zu erkennen. So hätten die zwei nachfolgend betrachteten Beispiele höchstwahrscheinlich durch einfache Tests gefunden werden können:

- Der Erstflug der Ariane 5 endete nach 40 Sekunden aufgrund einer Fehlfunktion der Software, die die Flugbahn der Rakete veränderte. Nach dem Abweichen vom vorgesehenen Kurs wurde die Rakete aus Sicherheitsgründen automatisch zerstört. Die Fehlfunktion wurde auf die Wiederverwendung von Softwarekomponenten der Vorgängerrakete Ariane 4 zurückgeführt. Da die Software im Rahmen von Ariane 4 problemlos funktionierte, wurde sie auf Ariane 5 übertragen, ohne dass gründliche Tests mit den zu erwartenden Ariane 5-Daten durchgeführt wurden. Aufgrund der besonderen Flugeigenschaften der Ariane 5 erhielt die Software für sie unerwartete Eingabedaten. Die Verarbeitung dieser Daten außerhalb des vorgesehenen Wartebereichs, d. h. des für die Ariane 4 typischen, führte zu einer Fehlfunktion, die die Rakete von ihrer Flugbahn ablenkte. [Majchrzak, 2012]
- Der Mars Climate Orbiter sollte die Klimabedingungen auf dem Mars untersuchen. Im September 1999 wurde der Orbiter bei dem Versuch, eine Umlaufbahn um den Mars zu erreichen, aufgrund von atmosphärischen Spannungen und starker Reibung zerstört. Geplant war, dass der Climate Lander in einer Höhe von etwa 150 km in die Umlaufbahn eintreten sollte. Stattdessen versuchte er jedoch, seine Umlaufbahn in 57 km Höhe zu beginnen, was sich als zu niedrig herausstellte. Dieser Fehler war auf ein Softwareproblem zurückzuführen: Ein Zulieferer für die Systeme des Climate Lander verwendete imperiale Einheiten anstelle des Internationalen Einheitensystems (SI). Da keine Umrechnung zwischen den Einheitensystemen vorgenommen wurde, wurden die imperialen Messungen fälschlicherweise als SI-Werte interpretiert, was zu fehlerhaften Berechnungen und ungewollten Kurskorrekturen führte. [Majchrzak, 2012]

Weitere Beispiele historischer schwerwiegender Fehler in softwarebasierten komplexen Systemen sowie deren Ursachen und Auswirkungen können u. a. in [Kopec & Tamang, 2007] nachgelesen werden.

Automatisierte und autonome Schiffe (AS) sind als komplexe, sicherheitskritische Systeme zu verstehen, die durch den kooperativen Betrieb von Software- und Hardwarekomponenten charakterisiert sind. Die zunehmende Integration von künstlicher Intelligenz erschwert eine umfassende Validierung dieser Systeme zusätzlich. AS sind somit eine Untergruppe automatisierter sicherheitskritischer cyberphysischer Systeme, zu denen auch unbemannte Luftfahrzeuge, führerlose Lieferroboter und kleinere autonome Seefahrzeuge zählen. Die Herausforderungen, die mit dem Beweis der korrekten Funktionalität von AS verbunden sind, sind daher zu Teilen analog zu den Herausforderungen, die in den anderen Untergruppen dieser breiteren Klasse automatisierter CPS bestehen. [Klikovits et al., 2024]

Im Folgenden werden zwei aktuell vielversprechende Ansätze für die Erprobung solcher fortschrittlichen Schiffe und den Beweis ihrer funktionalen Sicherheit logisch in die bisherigen Inhalte eingeordnet, zum Verständnis nötige Konzepte zuvor eingeführt und die bestehenden Herausforderungen herausgearbeitet. Begründet durch die evidenten Gemeinsamkeiten mit automatisierten Fahrzeugen anderer Verkehrsdomänen als sicherheitskritische CPS werden dabei auch hier gelegentlich Herausforderungen vorhandene Lösungsansätze zum Beweis von Funktionalität und Sicherheit dieser betrachtet.

7.1 VERIFIZIERUNG UND VALIDIERUNG

Die Prüfung stellt einen integralen Bestandteil des Entwicklungszyklus eines automatisierten maritimen Fahrsystems als sicherheitskritisches CPS dar, da der Beweis darüber, dass diese unter einer Vielzahl von Bedingungen zuverlässig und sicher operieren können, zwingend erbracht werden muss. Verifizierung und Validierung sind strukturierte Verfahren, die jeweils eine Vielzahl von konkreten Methoden und Aktivitäten rund um die Analyse und das Testen von softwarebasierten Artefakten umfassen. Durch die Anwendung dieser sollen die Qualität und Zuverlässigkeit gemessen werden. Beide haben ihren Ursprung in der Systemtechnik, die Software in einem ganzheitlichen Systemkontext bewertet, um diese im Hinblick auf alle Systemfunktionen sowie auf Hardware-, Benutzer- und andere Softwareschnittstellen zu analysieren und zu testen [Wallace & Fujii, 1989]. Die Richtlinie des *Vereins Deutscher Ingenieure* (VDI) *Entwicklungsmethodik für mechatronische Systeme* definiert beide Begriffe getrennt voneinander wie folgt:

Verifizierung: Im Allgemeinen ist mit *Verifizierung* (oder *Verifikation*) der Nachweis der Wahrheit von Aussagen gemeint. Übertragen auf technische Systeme ist hierunter die Überprüfung zu verstehen, ob eine Realisierung mit der Spezifikation übereinstimmt. Bei der Überprüfung der Gültigkeit eines Programms wird auch von der Programmverifikation gesprochen. Umgangssprachlich ist die Verifikation die Beantwortung der Frage: „Wird/wurde ein *korrektes* Produkt entwickelt?“ [VDI, 2206]

Validierung: Ursprünglich stellt die Validierung die Gültigkeitsprüfung einer Messmethode in der empirischen Sozialforschung dar. Das heißt, es wird die Frage beantwortet, inwieweit die Testresultate tatsächlich das erfassen, was durch den Test bestimmt werden soll. Übertragen auf technische Systeme ist hierunter die Prüfung zu verstehen, ob das Produkt für seinen Einsatzzweck geeignet ist und den gewünschten (Mehr-)Wert erzielt. Dabei fließen die Erwartungen von Fachexperten und Anwendern in die Bewertung ein. Die Validierung beinhaltet z. B. die Prüfung, ob die Beschreibung eines Algorithmus mit dem zu lösenden Problem übereinstimmt. Umgangssprachlich ist die Validierung die Beantwortung der Frage: „Wird/wurde das *richtige* Produkt entwickelt?“ [VDI, 2206]

Zusammengefasst kann gesagt werden, dass im Rahmen der Verifizierung versucht wird, zu zeigen, dass ein entwickeltes Artefakt den zuvor definierten Spezifikationen entspricht. Die Validierung soll hingegen eine Aussage darüber treffen, ob ein Artefakt seinen vorhergesehenen Zweck erfüllt. Eine beispielhafte Betrachtung eines Assistenzsystems zur aktiv eingreifenden Kollisionsvermeidung verdeutlicht den Unterschied: Wurde die Validierung erfolgreich abgeschlossen, ist davon auszugehen, dass die Spezifikationen – also die zuvor definierten Anforderungen an das System – vom System erfüllt werden. Eine solche Anforderung könnte z. B. sein, dass das Schiff niemals eine Ausweichroute wählt, die auf der linken Seite eines entgegenkommenden Schiffes liegt. Auch wenn alle Spezifikationen eingehalten werden, ist nicht automatisch sichergestellt, dass das entwickelte System auch seinen vorgesehenen Zweck erfüllt. Bei dem beispielhaft betrachteten Kollisionsvermeidungssystem könnte dies z. B. die Verringerung von Kollisionen im Vergleich zu Schiffen ohne dieses System sein. Geht man nun von dem Fall aus, dass die im Vorfeld definierten Anforderungen nicht die richtigen Anforderungen sind und/oder unvollständig oder ungenau sind, so wird das neue System erfolgreich validiert, aber nicht erfolgreich verifiziert werden können. Die Verifizierung bezieht sich somit ausschließlich auf die Funktionalität gemäß den Spezifikationen, die Validierung hingegen versucht, die Praxistauglichkeit zu beweisen.

Obwohl – oder gerade weil – Verifizierung und Validierung klar separierte Definitionen, Verfahrensinhalte und Ziele haben, kann der Nutzen maximiert durch eine synergetische Nutzung dieser Konzepte erreicht werden, indem *V&V* als eine integrierte Einheit betrachtet wird [Wallace & Fujii, 1989]. Daher wird V&V heutzutage im Kontext von Software-, System- oder Produktentwicklung in der Regel als eine integrierte Einheit verstanden, definiert und angewendet. Dabei umfasst V&V weit mehr Aktivitäten als das bloße Testen einer Neuentwicklung, denn das bloße Aus- oder Durchführen eines beliebigen Tests hat keine Aussagekraft über die Qualität des zu testenden Artefakts. Vielmehr müssen die Tests auf die individuellen konkreten Anforderungen des zu testenden Artefakts ausgerichtet sein und die Ergebnisse im Anschluss an die Durchführung ebenso individuell ausgewertet werden, um aussagekräftige Ergebnisse zu liefern. V&V beginnt somit bereits vor und endet nach den eigentlichen Testaktivitäten.

7.1.1 V&V-AKTIVITÄTEN IM V-MODELL

V&V-Aktivitäten sind in dem, in Abschnitt 6.4 eingeführten, V-Modell klassischerweise im rechten Drittel eingeordnet: Nachdem die Entwicklung des Systems fertiggestellt wurde, wird dieses zunächst schrittweise gegen die auf der linken Seite entstandenen Spezifikationen verifiziert und anschließend das integrierte System, den Gesamtprozess abschließend, validiert.

Dieser dreiteilige und streng getrennte Prozess der Planung, Umsetzung und Prüfung hat sich für die Entwicklung moderner, nichtdeterministischer und hochkomplexer Systeme, wie automatisierter und autonomer Fahrzeuge, als nicht mehr ausreichend erwiesen (vgl. [Koopman & Wagner, 2016]). Ein wesentlicher Grund dafür ist, dass V&V-Aktivitäten im Rahmen des V-Modells erst in einem späten und weit fortgeschrittenen Stadium in den Gesamtentwicklungsprozess integriert werden [Brinkmann, 2018]. Vor allem die mögliche operationale und verwaltungstechnische Eigenständigkeit der Komponenten des Fahrzeugs als Gesamtsystem (vgl. Abschnitt 6.3) kann dabei in Verbindung mit dem angestrebten und notwendigen emergenten Verhalten zu Fehlern oder zum Nicht-Erfüllen der Anforderungen an die emergenten Eigenschaften des Fahrzeugs führen, die nur ressourcenintensiv zu identifizieren und zu beheben sind. Nielsen et al. nennen die folgenden vier Gründe für ein Versagen eines System-of-Systems auf der obersten Systemebene als die häufigsten [Nielsen et al., 2015]:

- Das Gesamtsystem soll entscheidende nichtfunktionale emergente Eigenschaften besitzen. Nichtfunktionale Eigenschaften, insbesondere die hier relevante Eigenschaft der Sicherheit, sind nicht kompositionsfähig, d. h., die bewiesene Sicherheit aller Teilsysteme hat keine Aussagekraft über die Sicherheit des Gesamtsystems [Leveson, 2020; Leveson, 2012].
- Die Teilsysteme entsprechen häufig nicht vollständig ihren Spezifikationen, wenn sie das erste Mal vom Zulieferer für die erste Durchführung eines Integrationstests ausgeliefert werden. Dadurch kann für die anderen Teilsysteme eine Diskrepanz bezüglich der Kommunikation mit dem fehlerhaften System entstehen, was wiederum zu Fehlschlägen von Tests auf Ebene des integrierten Gesamtsystems führen kann.
- Sind die Spezifikationen nicht vollständig, dann treffen die Lieferanten der Teilsysteme ggf. selbstständig Annahmen über das Verhalten der anderen Teilsysteme und der Kommunikation mit diesen. Werden diese Annahmen nicht konsequent dokumentiert und kommuniziert, kann es vorkommen, dass diese während des Betriebs als integriertes System-of-Systems nicht erfüllt werden. Fehlerhaftes Verhalten des Teilsystems, dessen Annahmen nicht erfüllt werden, ist in der Regel die Folge.

- Das Scheitern von Tests auf Systemebene kann auf die unzureichende Erfassung von emergenten Eigenschaften während der Phase der Anforderungserhebung zurückgeführt werden.

Eine Möglichkeit, diesen Problemen zu begegnen, ist, die Kommunikation der Teilsysteme miteinander bereits während ihrer jeweiligen Entwurfs- und Entwicklungsphasen (vgl. Abbildung 17 und Abbildung 18) zu verifizieren und zu validieren. Aufgrund der verwaltungstechnischen Unabhängigkeit der einzelnen Teilsysteme eines Schiffes als integriertes System-of-Systems ist es aber im Allgemeinen nicht ohne erheblichen Mehraufwand möglich, die Entwicklungs- und Produktlebenszyklusaktivitäten zwischen ihnen zu synchronisieren. Es kann somit nicht gewährleistet werden, dass alle Systeme, die als Teil des übergeordneten Systems *Schiff* geplant sind, sich zu einem bestimmten Zeitpunkt im selben Entwurfs- oder Entwicklungszustand befinden. Somit ist die Durchführung gemeinsamer Tests der Teilsysteme, oder in frühen Entwurfsphasen erster Modelle dieser, während der Entwicklung nicht realistisch. Betrachtet man beispielhaft den Fall, dass als Brückenführungssoftware eine Standardsoftware ausgewählt wird, in welche lediglich einige kleinere Anpassungen integriert werden, für die Verarbeitung der Sensordaten auf einem modernen Schiff aber die Entwicklung einer Individualsoftware beauftragt wird, so wird schnell klar, wie unterschiedlich die Aufwände bei den beiden Zulieferern sein und wie stark sich die Entwicklungszyklen daher unterscheiden werden.

7.1.2 ENTWICKLUNGSBEGLEITENDE V&V

Diese Herausforderung kann mit Hilfe von strengen Interoperabilitätstestkampagnen angegangen werden: Im Voraus muss klar definiert sein, welche Daten die Systeme in welcher Struktur erwarten sowie bereitstellen. Sind diese In- und Outputs klar definiert, kann ein einzelnes Teilsystem simulativ getestet werden, indem es im aktuellen Entwicklungszustand in ein Modell eines Schiffes integriert wird, welches genau diese Daten in der erwarteten Struktur an das System übergibt. So kann vor der Integration in das Schiff als System-of-Systems zumindest geprüft werden, ob die Schnittstellenspezifikationen eingehalten wurden und das Teilsystem auf gegebene Inputs reagiert wie geplant. Offenbart sich durch solche simulative Tests bereits ein Fehlverhalten des Teilsystems, so kann dies frühzeitig und somit kosteneffizient behoben werden, bevor die Integrationstests am Ende des Gesamtentwicklungszyklus des Schiffes durchgeführt werden.

Folglich müssen Teilsysteme und Komponenten des Fahrzeug-Gesamtsystems bereits vor Beginn ihres eigenständigen Lebenszyklus eindeutig identifiziert und spezifiziert werden. Während der jeweiligen Erstellungsphase muss die Korrektheit bereits entwicklungsbegleitend sichergestellt werden, bevor die Gesamtentwicklung mit der stufenweisen Integration der Teilsysteme und Komponenten fortgesetzt wird. In Abbildung 19 wurde zu diesem Zweck jeder Entwicklungsaktivität der Entwurfsphase sowie der Umsetzungsphase eine individuelle Feedbackschleife zugewiesen. Diese deuten an, dass die individuellen Artefakte der jeweiligen Aktivitäten in Form von abnehmend abstrakten Modellrepräsentationen bereits während des Entwurfes und der Umsetzung durch Validierungs- und Verifizierungstätigkeiten geprüft werden. Die dabei identifizierten Fehler werden in die jeweilige Stufe zurückgeführt, um entsprechende Anpassungen zur Behebung dieser vorzunehmen. Diese Schleifen können jeweils mehrfach durchlaufen werden, um zum einen entwicklungsunterstützend (vgl. das Konzept des Test-Driven-Development [Mellor, 2023]) als auch absichernd vor dem Übergang in die nächste Aktivität (d. h. klassisch) wirken zu können. Eine solche kontinuierliche V&V über den gesamten Lebenszyklus

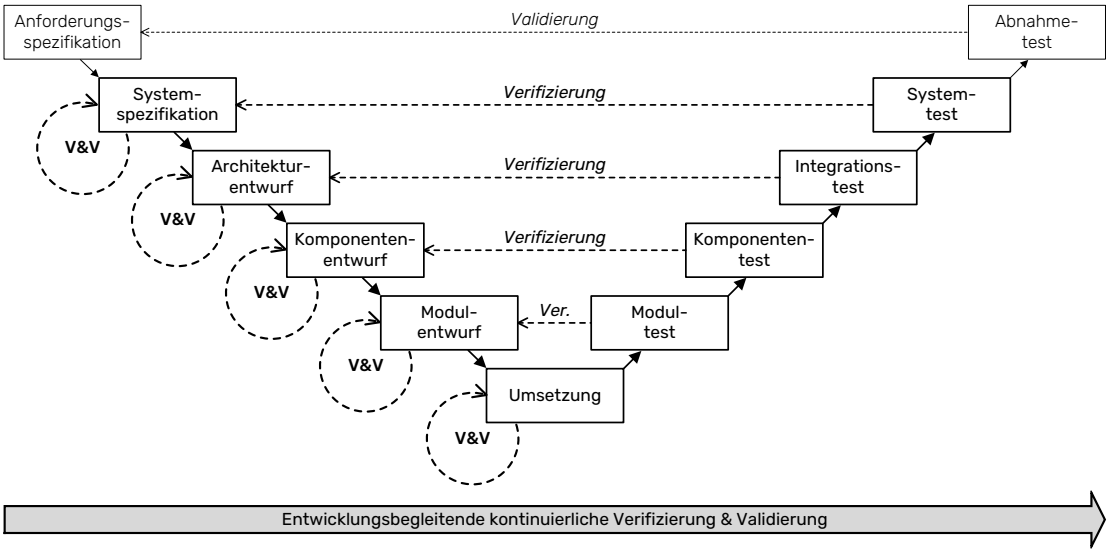


Abbildung 19: Durch Integration individueller V&V-Feedback-Schleifen in die einzelnen Entwurfs- und Umsetzungsphasen des Entwicklungsprozesses von automatisierten Fahrzeugsystemen angepasstes V-Modell (Idee und Abbildung basierend auf den in [Koopman und Wagner, 2016] und [Brinkmann, 2018] vorgeschlagenen Prozessen). Phasen außerhalb des Betrachtungsumfangs dieser Arbeit sind grau dargestellt.

gewährleistet die frühzeitige Erkennung von Fehlern und schafft so die Voraussetzung für die kostengünstige Beseitigung dieser Fehler (vgl. [Karthaus et al., 2021; Tibba et al., 2016]).

Ein vergleichbarer Ansatz, bei dem die Idee des von Thomke & Fujimoto unter dem Namen *Front-Loading* vorgeschlagenen Konzeptes einer frühen Problemlösungsperspektive für das Management von Produktneuentwicklung [Thomke & Fujimoto, 2000] in das V-Modell zur Systementwicklung integriert wurde, wurde von Brinkmann unter dem Namen *Frontloading der V&V* skizziert [Brinkmann, 2018]. Auch in diesem Vorschlag sollen Spezifikationen und Anforderungen bereits in der ersten Phase des V-Modells, der Gestaltung, verifiziert und validiert werden. Dass die integrierten V&V-Aktivitäten in Verbindung mit den einzelnen Entwurfs- und Umsetzungsschritten jeweils Feedback-Schleifen bilden, wird dort aber nicht explizit hervorgehoben. Das Konzept des Front-Loadings wird mit Bezug auf die Entwicklung moderner Straßenfahrzeuge auch in [Karthaus et al., 2021] beschrieben. Der Fokus der Autoren liegt dort auf dem Entwurf von Testfällen zur Verwendung im Produkterprobungsprozess, und die Integration in den Entwicklungsprozess wird nicht explizit betrachtet.

Andere Arbeiten gehen bzgl. des Umdenkens bei der Gestaltung der Entwicklungsprozesse noch einen Schritt weiter. So argumentieren Mallozzi et al., begründet durch die enorme Zunahme von Software in Fahrzeugen, dass sich die Entwicklung vom starren V-Modell hin zu flexibleren Methoden entwickelt hat. Dabei werden explizit agile Methoden erwähnt, die in den vergangenen Jahren an vielen Stellen für ein Umdenken in der angewandten Softwareentwicklung gesorgt haben, bei denen Iterationen deutlich häufiger sind und somit häufig ein kurzfristiges Feedback möglich ist. [Mallozzi et al., 2019]

Dem konträr gegenüber stehen die etablierten Normen für die Entwicklung von Fahrzeugen und in diese integrierte Software, wie die ISO 17894 [ISO, 17894:2005]. Vollständig agile Methoden sind zwar in der Lage, schnelle Ergebnisse hervorzubringen und die Möglichkeit zu bieten, früh und effizient auf Softwarefehler zu reagieren, für sicherheitskritische Systeme erscheinen sie aber zu unstrukturiert, um den für Fahrzeuge hohen Sicherheitsanforderungen und -vorschriften gerecht zu werden. Da vollständig agile Methoden somit keine realistische Chance auf Anwendbarkeit bei der Entwicklung automati-

sierter Fahrzeuge als sicherheitskritische CPS haben, stellt der Versuch, das etablierte V-Modell zwar beizubehalten, aber durch die Integration von Feedbackloops in den einzelnen Schritten agiler zu machen, einen tragbaren Kompromiss dar. Das V-Modell bietet dabei den entscheidenden Vorteil, dass zu Beginn bereits vollständige Anforderungen festgehalten werden müssen – somit auch an sicherheitskritischen Funktionalitäten, welche auf der rechten Seite als Grundlage für die Testphase genutzt werden. Zusammen mit einer vollumfänglichen Dokumentation aller (Teil-)Ergebnisse kann so eine eindeutig nachvollziehbare Aussage über die Sicherheit erreicht werden. Die kontinuierliche Integration der V&V-Feedback-Schleifen fügt zusätzliche Vorteile hinzu [Wallace & Fujii, 1989], welche denen der agilen Entwicklung ähneln:

- Risikoreiche Fehler werden frühzeitig aufgedeckt, sodass das Designteam Zeit hat, eine umfassende/kosteneffiziente Lösung zu entwickeln, anstatt eine behelfsmäßige/kostenintensive Lösung zu erzwingen, um die Entwicklungsfristen einzuhalten.
- Statt ausschließlich das Gesamtsystem werden auch die Teilergebnisse bereits im Hinblick auf die Systemanforderungen evaluiert.
- Das Management (sowohl das für das konkrete Teilsystem verantwortliche als auch das des Fahrzeugs als Gesamtsystem; vgl. Abbildung 15, Abbildung 18) erhält kontinuierliche und umfassende Informationen über die Qualität und den Fortschritt der Entwurfs- und Entwicklungsarbeit.
- Sie gibt dem Management/Auftraggeber eine schrittweise Vorschau auf die Systemleistung und ermöglicht, frühzeitig Anpassungen vorzunehmen.

7.2 SZENARIOBASIERTES TESTEN

Moderne Technologien zur automatisierten Führung, Navigation und Steuerung von Schiffen stellen aufgrund der in Abschnitt 6 angesprochenen inhärenten Systemkomplexität, der nötigen funktionalen Zuverlässigkeit und der hohen Sicherheitsanforderungen anspruchsvolle Anforderungen an die Verifizierung und Validierung (V&V). Diese Herausforderungen werden durch das Konzept des offenen System-of-Systems (SoS) und der Integration von künstlicher Intelligenz weiter verschärft. Vor allem die erwünschte Emergenz der Teilsysteme führt dazu, dass die bisher häufig üblichen isolierten Prüfungen für eine hinreichende V&V nicht mehr ausreichen. Fehlerhafte oder ausbleibende emergente Eigenschaften und Fehler in der Kommunikation der Teilsysteme lassen sich ausschließlich im Rahmen von Tests des Gesamtsystems identifizieren [Brinkmann et al., 2018].

Folglich ist die Anwendung neuer Testmethoden im Bereich des Systems Engineering von automatisierten Fahrzeugen notwendig. Ein vielversprechender und u. a. in der Domäne des Straßenverkehrs bereits teilweise Anwendung findender Ansatz ist das sogenannte szenariobasierte Testen (vgl. Projekte wie [DLR, 2019] und [Leitner et al., 2019]). Dieser Abschnitt arbeitet das grundlegende Thema des Testens von softwarebasierten Systemen auf, um anschließend einen Überblick über den szenariobasierten Ansatz zu geben, die Bedeutung der zentralen Begriffe im Detail zu festigen und die Herausforderungen bei der Anwendung dieses jungen Ansatzes aufzuzeigen.

7.2.1 TESTEN SOFTWAREBASIERTER SYSTEME

Das Testen als eigenständige und relevante Teil-Aktivität der Softwareentwicklung hat erstmals in den 1970er Jahren langsam an Bekanntheit gewonnen. Eines der ersten umfassenden Bücher über Softwaretests wurde 1973 veröffentlicht und basiert auf den Beiträgen des 1972 stattgefundenen *Computer*

Program Test Methods Symposiums [Hetzel, 1973]. Wenig später schrieb Myers das Buch *The Art of Software Testing* [Myers, 1979], das nach seiner Veröffentlichung 1979 einen bedeutenden Einfluss auf die gelebte Praxis ausübte. Die durch Corey Sandler und Tom Badget aktualisierte und 2012 veröffentlichte dritte Edition [Myers et al., 2012] ist bis heute ein relevantes Grundlagenwerk. Myers wird daher häufig die Gestaltung der bis heute gültigen Grundprinzipien des Testens von Software und softwarebasierten Systemen zugeschrieben.

„To test a program is to try to make it fail."
[Meyer, 2008]

Ineffektive Testaktivitäten und daraus resultierend schlechte oder fehlerbehaftete softwarebasierte Systeme sind häufig verursacht durch eine falsche Vorstellung davon, was das Ziel des Testens eigentlich ist. Aussagen wie *„Testen ist der Prozess des Nachweises, dass keine Fehler vorhanden sind."* oder *„Der Zweck des Testens ist es, zu zeigen, dass ein Programm seine beabsichtigten Funktionen korrekt ausführt."* betrachten das Problem aus der falschen Richtung. Vielmehr muss beim Testen davon ausgegangen werden, dass immer Fehler vorhanden sind. Das verfolgte Ziel sollte dann sein, möglichst viele dieser Fehler zu identifizieren und zu beheben. [Myers et al., 2012]

Majchrzak definiert Testen als den Prozess des Betriebs eines (Software-)Systems oder einer (Software-)Komponente, dem *Testobjekt*, unter bestimmten Bedingungen bei gleichzeitiger Beobachtung oder Aufzeichnung der Ergebnisse und einer Bewertung bestimmter Aspekte des Systems oder der Komponente [Majchrzak, 2012]. Da auch hier der deutschsprachige Begriff *Fehler* ähnlich viele Bedeutungen hat wie der in Abschnitt 2 betrachtete Begriff *Sicherheit*, werden nachfolgend ebenfalls die eindeutigeren englischsprachigen Begriffe herangezogen und die daraus entstandenen deutschen Äquivalente eingeführt: Durch Tests sollen Fehler in der Software (engl.: defects), d. h. teilweise inkorrekte Umsetzungen dieser, aufgedeckt werden. Diese *Fehlerzustände* werden in der Regel durch menschliche Fehler (engl.: errors), wie der Eingabe einer falschen Zahl, Missverständnisse bzgl. Programmspezifikation, Unkenntnisse in Bezug auf die verwendete Programmiersprache oder Unvollständigkeit der Entwicklungsprozessdokumentation, verursacht [Majchrzak, 2012]. Diese Fehler während des aktiven Umsetzens werden als *Fehlerhandlungen* bezeichnet. Wird die Konsequenz eines durch eine *Fehlerhandlung* verursachten *Fehlerzustands* während des Betriebs/der Ausführung des betroffenen (Software-)Systems/der betroffenen (Software-)Komponente als Abweichung von der Spezifikation sichtbar, so handelt es sich um eine *Fehlerwirkung* (engl.: failure). Auch eine Abweichung von den Nutzererwartungen kann als *Fehlerzustand* klassifiziert werden, da diese vom Nutzer als *Fehlerwirkung* wahrgenommen werden kann und auf Fehlentscheidungen oder unsaubere Prozesse zurückzuführen ist [Majchrzak, 2012]. Die Beziehungen dieser drei, für das Verständnis von Fehlern und ihrer Entstehung essenziellen Begriffe sind in Abbildung 20 unterstützend grafisch veranschaulicht.

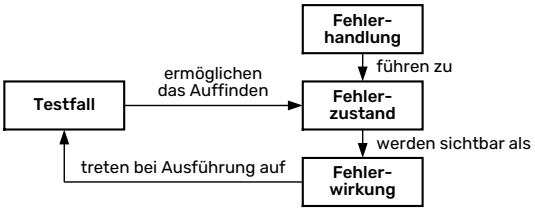


Abbildung 20: Relevante Begriffe zur Beschreibung von Fehlern, ihren Ursachen und Auswirkungen softwarebasierter Systeme. (Angelehnt an [Majchrzak, 2012]; deutschsprachige Begriffe übernommen aus [Spillner und Linz, 2012];)

Im einleitenden Teil I wurde bereits angesprochen, dass statische Testmethoden wie die Modellprüfung (engl.: model checking) nicht geeignet sind, um moderne automatisierte Fahrzeuge oder Teilsysteme dieser zu testen. Modellprüfung ist ein mächtiges Werkzeug, um die Korrektheit eines Modells/eines Systems/einer Software gegen eine vorliegende Spezifikation zu verifizieren (vgl. u. a. [Bérard et al., 2001]). Mithilfe von Modellprüfung hätte etwa der einleitend erwähnte Absturz der Ariane 5 verhindert werden können: Die Ursache war hier ein Fehlerzustand, der durch einen Speicherüberlauf bei der Umwandlung einer 64-Bit-Gleitkommazahl in einen 16-Bit-Ganzzahlwert mit Vorzeichen auftrat. Daraus resultierte ein Operandenfehler. Die Umwandlung der Zahl schlug somit fehl, weil die horizontale Geschwindigkeit, die von der Plattform gemessen wurde, höher war als erwartet, da sich der frühe Teil der Flugbahn der Ariane 5 bezüglich der horizontalen Geschwindigkeit stark von der erwarteten Flugbahn (der Flugbahn der Ariane 4) unterschied. Model-Checking kann solche Umwandlungsfehler entdecken, wenn entsprechende Grenzen (Minimum, Maximum etc.) im Rahmen der Spezifikation bedacht und beschrieben wurden. Modellprüfverfahren können dabei für viele Testobjekte deutlich effizienter sein als die manuelle Durchführung von Tests oder die Simulation einer Vielzahl von Testfällen zur Laufzeit und können, bei bekannter Struktur des Datenflusses innerhalb der Software, auch die Abwesenheit von Fehlern beweisen. Zur Überprüfung von Testobjekten, die auf Modellen selbstlernender KI-Verfahren aufbauen, funktioniert dieser Ansatz deshalb nicht mehr, aufgrund der bisher nicht vorhandenen Möglichkeit, ein umfassendes Verständnis über das resultierende gelernte Modell zu schaffen. Zusätzlich ist die Umgebung, in welcher solche KI-basierten Fahrsysteme eingesetzt werden, höchst dynamisch und nicht deterministisch, d. h., der Zustandsraum, in welchem das zu prüfende System eingesetzt werden soll, ist nicht fest definiert und nicht endlich.

Eine adäquate Testmethode muss daher im Bereich der dynamischen Testmethoden gesucht werden (vgl. Abbildung 21). Statische Testmethoden sehen keine Ausführung der Software vor und mithin ebenfalls keine Eingabe von Eingangs- oder Störgrößen sowie keine Untersuchung von Ausgabegrößen. Stattdessen werden auf Basis des Wissens über das zu testende System die möglichen Daten- und Kontrollflüsse abgebildet und anschließend mit der vorliegenden Spezifikation abgeglichen [Majchrzak, 2012; Spillner & Linz, 2012]. Auf der anderen Seite des Spektrums befinden sich die dynamischen Testmethoden: Hier werden dem Testobjekt Eingangsgrößen und ggf. auch Störgrößen (zusammengefasst als *Testdaten* bezeichnet) zur Verfügung gestellt und zur Ausführung gebracht [Spillner & Linz, 2012]. Die durch die interne Transformation dieser durch das Testobjekt erzeugten Ausgangsgrößen

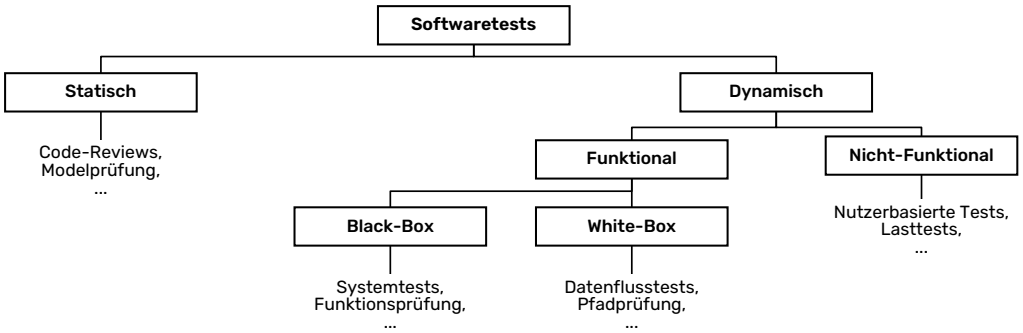


Abbildung 21: Klassifikation von Methoden zum Testen von Software(-basierten) Systemen und Komponenten. Die dargestellten Kategorien sind in der Realität nicht alle scharf voneinander abgegrenzt. So kann ein funktionaler Test ebenso nichtfunktionale Aspekte überprüfen. Auch im Grenzbereich zwischen Blackbox-Tests und Whitebox-Tests hat sich eine weitere Kategorie etabliert: der Greybox-Test.

werden anschließend in der Regel mit den auf Basis der vorliegenden Spezifikation erwarteten Ausgangsgrößen verglichen. Üblicherweise werden Testfälle dabei so gewählt, dass nach der Durchführung aller Testfälle möglichst jeder Aspekt der Spezifikation überprüft wurde und/oder möglichst viele Bereiche im Testobjekt mindestens einmal aktiv genutzt wurden. Als Maß für Letzteres werden für reine Software-Testobjekte häufig die Anweisungsabdeckung oder Zweigabdeckung genutzt. Ein Testfall umfasst dabei immer Eingangsgrößen, erwartete Ausgangsgrößen sowie Randbedingungen, welche zu Beginn der Durchführung des Tests erfüllt sein müssen [Spillner & Linz, 2012].

Innerhalb der Menge der dynamischen Tests kann weiter eine Klassifikation in die Teilmengen der funktionalen sowie nicht funktionalen Tests unternommen werden. Nicht-funktionale Tests überprüfen dabei Aspekte wie die Nutzerfreundlichkeit, Ausführungsgeschwindigkeiten, Reaktionszeiten und das Verhalten unter Last. Diese Arbeit beschäftigt sich mit der Prüfung der funktionalen Sicherheit automatisierter Fahrzeuge und Fahrfunktionen sowie der Teilsysteme von erstgenanntem. Entsprechende Testmethoden sind daher in der Menge der funktionalen Testmethoden zu suchen. Charakterisiert durch das Wissen, das bei Erstellung des Tests über die inneren Abläufe des Testobjektes vorhanden ist, werden funktionale Tests weiter in die Kategorien der sogenannten Blackbox- und Whitebox-Tests eingeteilt. Während im Rahmen von Whitebox-Tests detailliertes Wissen über die Strukturen und Abläufe innerhalb des Testobjektes vorhanden ist und genutzt wird, wird das Testobjekt bei Blackbox-Tests ohne Wissen über die Transformation selbst in die Tests integriert (engl.: blackbox; dt.: schwarze Kiste; Analogie für den verwehrten Blick in das Innere) (vgl. Abbildung 12).

7.2.2 ZUSTANDSRÄUME KOMPLEXER SYSTEME

Eine der größten Herausforderungen bei der Prüfung komplexer Systeme ist die als *State Space Explosion* bezeichnete Tatsache, dass die Anzahl möglicher Systemzustände bei Steigerung der Komplexität des zu prüfenden Systems nicht linear, sondern deutlich stärker wächst und sich für die zu prüfenden Systeme extrem große zu prüfende Zustandsräume ergeben [Pecheur, 2000]. Um die korrekte Funktionalität zu beweisen, wäre das als offensichtlich erscheinende Vorgehen, jede mögliche Permutation von Zuständen des Systems, der Teilsysteme und Komponenten sowie der Systemumgebung und Eingangsgrößen und Störgrößen zu testen. Ein solches Vorhaben wird sich in der Realität jedoch als nicht realisierbar darstellen: Selbst eine scheinbar einfache Softwarekomponente kann Tausende mögliche Eingabe- und Ausgabekombinationen aufweisen, deren Testung zeitlich unpraktisch und im kommerziellen Umfeld finanziell untragbar wäre [Myers et al., 2012]. Der Nachweis absoluter Fehlerfreiheit ist somit aktuell nicht möglich [Spillner & Linz, 2012].

Um sich der möglichen Ausmaße der Zustandsräume und somit der Relevanz der State Space Explosion von komplexen Systemen bewusst zu werden, ist es aufschlussreich, einen Blick auf die Zustandsräume von trivialen Systemen zu werfen. Diese weisen oft bereits eine überraschend große Anzahl von möglichen Zuständen auf. Betrachtet man etwa ein kleines Programm, das zwei ganze Zahlen als Eingangssignale entgegennimmt und deren Summe zurückgibt, so existieren, unter der Annahme, dass die zugrunde liegenden Eingabe- und Ausgabe Funktionalitäten korrekt sind, bei Nutzung von 32-Bit-Datentypen bereits $2^{32} \cdot 2^{32} = 2^{64}$ mögliche Ausprägungen der Eingangssignale. Bei 64-Bit-Ganzzahlen wären es bereits 2^{128} $\left(\approx 3{,}4 \cdot 10^{38}\right)$ Möglichkeiten. [Majchrzak, 2012]

Als ein weiteres, etwas greifbareres Beispiel kann das zugrundeliegende System des Philosophenproblems (engl.: dining philosopher problem) betrachtet werden. Das Philosophenproblem ist ein klassi-

sches Fallbeispiel aus dem Bereich der theoretischen Informatik und wird zumeist genutzt, um das Problem der Nebenläufigkeit und die Gefahr der Verklemmung von ressourcennutzenden Prozessen zu veranschaulichen, und wurde zuerst 1971 von Dijkstra vorgestellt: Es sitzen n Philosophen $P = (p_1, \ldots, p_n)$ an einem runden Tisch, in dessen Mitte eine große Schüssel Spaghetti steht. Philosophen gehen auch an dem Tisch ihrem Tagesgeschäft nach und denken über schwerwiegende Probleme nach. Sobald sie Hunger bekommen, unterbrechen sie den Gedankenprozess allerdings und wollen essen. Zwischen jedem Paar benachbarter Philosophen befindet sich eine Gabel, sodass jeder Philosoph eine Gabel zu seiner Linken und eine Gabel zu seiner Rechten hat. Es wird angenommen, dass die Spaghetti sehr schwer zu essen sind, sodass ein Philosoph mit zwei Gabeln essen muss. Jeder Philosoph kann nur die Gabeln zu seiner unmittelbaren Linken und zu seiner unmittelbaren Rechten benutzen, wobei er zuerst nach der linken Gabel greift und erst, wenn er diese ergattern konnte, nach der Gabel zu seiner Rechten greift. Sobald ein Philosoph eine Gabel in jeder Hand hält, beginnt er unmittelbar zu essen. Jeder Philosoph befindet sich also stets in einem Zustand $z_{p_n} \in Z_P = \{DENKEN, HUNGER, LINKS, ESSEN\}$ (vgl. Abbildung 22a). Betrachtet man das beschriebene Fallbeispiel als System, so stellen die einzelnen Philosophen die Systemkomponenten des Gesamtsystems dar.

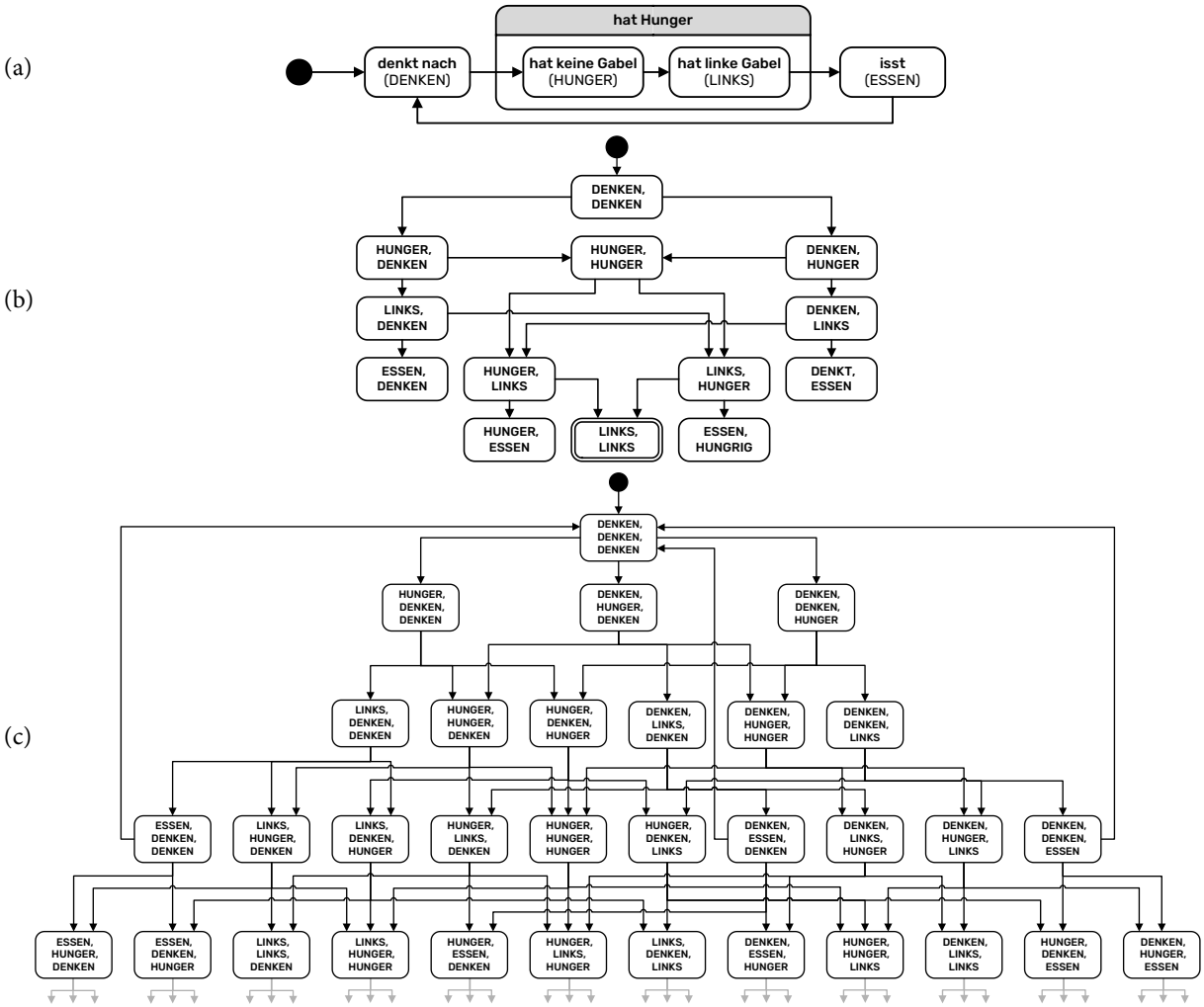


Abbildung 22: Mögliche Zustände des Philosophenproblems mit (a) einem, (b) zwei und (c) drei Philosophen, die jeweils einen der vier Zustände DENKEN, HUNGER, LINKS, ESSEN annehmen können. Dabei bedeutet LINKS, dass das linke Essbesteck genommen wurde und ESSEN, dass sowohl das Essbesteck zur Linken als auch das zur Rechten gehalten wird. Zur Verbesserung der Übersichtlichkeit sind die meisten rückführenden Zustandsübergänge nicht dargestellt. Aus gleichem Grund ist in (c) der Zustandsraum nach unten offen dargestellt.

Da sich der Systemzustand durch die Gesamtheit der Zustände der Systemkomponenten ergibt, wird dieser somit durch ein n-Tupel $Z_S = \left(z_{p_1}, z_{p_2}, \dots, z_{p_n}\right)$ beschrieben. Die Anzahl der möglichen Zustände des Gesamtsystems ergibt sich also unmittelbar aus der Anzahl teilnehmender Systemkomponenten (Philosophen) n und der Anzahl möglicher Zustände jeder dieser Komponenten-Philosophen $|Z_P|$. Einige Kombinationen sind aufgrund der gemeinsamen Ressourcennutzung als Kern des Problems nicht zulässig und sind daher auch nicht Teil der Menge der möglichen Systemzustände: Hat Philosoph *n+1* bereits die von ihm rechts platzierte Gabel genommen, er befindet sich also im Zustand *ESSEN*, so kann Philosoph *n* nicht auf die von ihm links platzierte Gabel zugreifen, d. h., nicht in den Zuständen *LINKS* oder *ESSEN* sein. Die Anzahl möglicher Systemzustände $|Z_S|$ liegt für n Philosophen mit jeweils $|Z|$ Zuständen somit im Bereich $|Z_P - 1|^n < |Z_S| < |Z_P|^n$. Für $n = 2$ und $n = 3$ sind die jeweiligen Zustandsräume in Abbildung 22b und c vereinfacht dargestellt.

Im Allgemeinen kann gesagt werden, dass die Größe des Zustandsraums eines Systems tendenziell exponentiell zur Zunahme der Anzahl der Systemkomponenten und der Zustandsvariablen des Systems (vgl. Abschnitt 6.3) wächst. Die Basis dieser Exponentialfunktion hängt von der Anzahl der lokalen Attribute der Komponenten ab und der Exponent wird in der Regel durch die Anzahl der Komponenten selbst bestimmt. Zusätzlich beeinflusst der Grad der Kopplung der Systemkomponenten, d. h. das Ausmaß, in dem die lokalen Zustände von Komponenten durch die lokalen Zustände anderer Komponenten bestimmt werden, die resultierende Größe des Zustandsraums. [Valmari, 1998]

In den zuvor besprochenen Beispielen sind die resultierenden Zustandsräume aufgrund der Einfachheit der Systemstruktur und der engen Kopplung zwischen den Systemkomponenten überschaubar: Im aufgezeigten Dining Philosopher Problem ist jeder Philosoph über die Ressource eines einzelnen Essbestecks direkt mit jedem seiner direkten Nachbarn gekoppelt, wodurch das kombinatorische Wachstum begrenzt wird. In realistischen Szenarien mit komplexeren Systemen, insbesondere solchen, die eine Verschachtelung weiterer Teilsysteme enthalten können, wird das Problem der State Space Explosion jedoch erheblich relevanter. Gründe dafür sind u. a. die zunehmende Nebenläufigkeit, die losere Kopplung der Komponenten, die umfangreicheren Funktionalitäten und die komplexeren inhärenten Abläufe. Eine Kombination dieser Faktoren kann zusätzlich zum exponentiellen Wachstum des Zustandsraums zu einer starken kombinatorischen Zunahme der Anzahl der Zustände beitragen.

Sollen offene Systeme (vgl. Abschnitt 6.3), wie auch automatisierte Fahrzeuge es sind, getestet werden, müssen zusätzlich zu den Systemkomponenten ebenfalls die Eingangs- und Störsignale betrachtet werden. Diese wirken auf das System und beeinflussen es. Um sicherzustellen, dass das zu testende System in allen Ausprägungen der Umgebung und der durch sie induzierten Signale zuverlässig funktioniert, sind diese ebenfalls als Teil des Zustandsraums zu betrachten. Da die Umgebung eines automatisierten Fahrzeugs ebenfalls ein System darstellt (ein Verkehrssystem), verhält sich die Größe der Menge an möglichen Systemzuständen hier analog zum betrachteten exponentiellen Wachstum beim zu testenden System. Die Menge der für eine vollständige Testabdeckung zu testenden System-Zustands-Tupel wächst somit zusätzlich um die Menge der möglichen Umwelt-Zustands-Tupel. Sollen alle Kombinationen möglicher Ausgangszustände des Systems und möglicher Ausgangszustände der Umwelt getestet werden, so können das zu testende System und seine Umwelt vereinfachend als ein einziges System gesehen werden, wodurch der exponentielle Anstieg noch steiler wird, da der Exponent um die Anzahl der Komponenten der Umwelt erhöht wird.

7.2.3 SZENARIOBASIERTER ANSATZ

Bei der Wahl einer Testmethodik zur Überprüfung von automatisierten Fahrfunktionen ist zusätzlich zu bedenken, dass die stark gestiegene Komplexität von Fahrzeugen sowie die analog dazu gestiegene Komplexität und Vernetzung der Teilsysteme u. a. dazu führen, dass die Funktionsgrenzen verschwimmen und Funktionstests eines einzelnen, isolierten Assistenzsystems somit nicht mehr ausreichen [Brinkmann et al., 2017]. Vielmehr muss das Gesamtfahrzeug als System-of-Systems (SoS) betrachtet werden und alle Teilsysteme oder repräsentative Substitutionen dieser müssen aufgrund der angestrebten emergenten Eigenschaften des SoS am Testlauf teilnehmen.

Vor allem mit Blick auf eine großflächige Einführung und den kommerziellen Vertrieb hochautomatisierter Fahrzeuge offenbart sich hier eine Hürde, die zunächst unüberwindbar scheint: die Sicherstellung der korrekten Funktionalität in allen Situationen, denen das Fahrzeug in Zukunft ausgesetzt sein könnte. Die Abdeckung aller möglichen System- und Umweltzustände sowie Kombinationen dieser (vgl. Abschnitt 7.2.2) im Rahmen von Testfahrten in der realen Welt erfordert einen unverhältnismäßig hohen Aufwand an Zeit und Ressourcen [Gelder & Op den Camp, 2021] (konkrete Beispiele und Zahlen dazu wurden bereits in Abschnitt 2 Absatz 7 genannt). Um diese Herausforderungen zu bewältigen, ist die Suche nach geeigneten Methoden für die Prüfung der funktionalen Sicherheit hochautomatisierter Fahrzeuge in den vergangenen Jahren weiter in den Fokus des Interesses gerückt. Ein Ansatz, der verspricht, den Gesamttestaufwand bei annähernd gleichbleibender Aussagekraft bzgl. der Sicherheit zu reduzieren, ist das szenariobasierte Testen [Ponn et al., 2020]. Der Ansatz basiert auf der Annahme, dass es Testfälle gibt, die mögliche Fehlerwirkungen effektiver sichtbar machen können als andere und somit eine höhere Aussagekraft bzgl. der funktionalen Sicherheit bieten [Fremont et al., 2020]. Um den Gesamttestaufwand möglichst gering zu halten, liegt es daher nahe, solche Testfälle vorab gezielt zu identifizieren, jeweils als Testfall zu formulieren und anschließend ausschließlich diese Testfälle durchzuführen, um insgesamt sowohl effektiv (hohe Aussagekraft) als auch effizient (geringer Testaufwand) die zu prüfenden Funktionalitäten zu testen [Klikovits et al., 2024].

Bezüglich des konkreten Ziels des Einsatzes szenariobasierter Tests finden sich in den aktuellen Forschungsbemühungen teils unterschiedliche Ansichten. Porres et al. schreiben in [Porres et al., 2020a], dass das Ziel darin besteht, *"[…] eine große Anzahl von Szenarien zu bewerten, um diejenigen zu finden, in denen die KI-Agenten nicht die erwartete Leistung erbringen"*[22]. Die Aussage, dass eine große Anzahl von Szenarien bewertet werden muss, ist zwar nicht gänzlich falsch, wird aber in der Regel nicht als Ziel des szenariobasierten Testprozesses angesehen. Vielmehr ist das Ziel, eine möglichst kleine Menge – welche durchaus noch von beträchtlicher Größe sein kann – Szenarien zu testen, die möglichst aussagekräftig bzgl. der korrekten Funktionalität des zu testenden Systems sind. Das von Porres et al. definierte Ziel ist eher in den Vorarbeiten für das eigentliche szenariobasierte Testen einzuordnen: Es muss eben jene Menge zu testender Szenarien identifiziert oder konstruiert werden, um als Eingabe für die eigentlichen Testläufe zu dienen (vgl. Abschnitt 4). Anhand der unglücklichen Formulierung der genannten Arbeit lässt sich darüber hinaus eine weitere generelle Anforderung an szenariobasierte Testmethoden erkennen. Würde szenariobasiertes Testen nach der o.g. Definition angewendet werden, würde sich das Problem ergeben, dass kein definiertes Ende der Testphase existiert. Die erfolgreiche Durchführung einer Menge von Testfällen beliebiger Größe reicht nicht aus, um eine Aussage über die

[22] Übersetzt aus dem Englischen; Auslassung;

korrekte Funktionalität des zu testenden Systems treffen zu können. Wird eine vordefinierte Menge Szenarien mit einer zuvor über Metriken oder Methoden wie der Kritikalität oder der Kombination von Randfällen bestimmten Aussagekraft als Eingabe für die eigentlichen Testläufe genutzt, hat die Testphase hingegen ein definiertes Ende und es kann eine begründete und nachvollziehbare Aussage bzgl. der funktionalen Sicherheit des zu testenden Systems getroffen werden. In dieser Arbeit wird daher die u. a. in [Fremont et al., 2020; Neurohr et al., 2020; Ponn et al., 2020; Bock et al., 2018] ebenfalls zu findende Auffassung vertreten, dass das Ziel szenariobasierten Testens die Reduzierung des Testaufwands durch die Reduzierung der Testfälle auf aussagekräftige Szenarien ist.

Ein wesentlicher Unterschied zwischen niedriger und hoher Automatisierung von Fahrzeugen (vgl. Abschnitt 6.1 für einen Überblick über einige gängige Hierarchien zur Klassifizierung automatisierter Fahrzeuge nach ihrem Automatisierungsgrad) liegt in der deutlich erhöhten Anzahl von Szenarien, die identifiziert, definiert und getestet werden müssen. Diese sind nötig, um auch für Systeme eines höheren Automatisierungsgrades eine verlässliche Aussage über ihre Zuverlässigkeit treffen zu können. [Bagschik et al., 2018]

7.2.4 SZENARIOBEGRIFF

Der Fokus des szenariobasierten Ansatzes liegt, entsprechend der Bezeichnung, auf der Verwendung von Szenarien als Fundament für die Durchführung von Tests. In der wissenschaftlichen Literatur existieren diverse Definitionen dessen, was genau die Inhalte eines *Szenarios* sind und wofür es eingesetzt werden kann. Viele dieser Definitionen sind zudem nicht direkt in ihrer präsentierten Form für den Einsatz als Basis szenariobasierten Testens automatisierter Fahrzeuge geeignet [Steimle et al., 2021]. Die ISO/IEC/IEEE-Norm für das Testen von Software im Rahmen von Software- und System-Engineering Prozessen definiert zum Beispiel zwar in [ISO/IEC/IEEE, 29199-1:2022] und [ISO/IEC/IEEE, 29199-4:2021] den Begriff *scenario testing* und in [ISO/IEC/IEEE, 29199-1:2022] den Begriff *test scenario*, bleibt dabei aber zu oberflächlich, um hier Anwendung zu finden. Zusätzlich setzt [ISO/IEC/IEEE, 29199-4:2021] ein Szenario mit einer vordefinierten Abfolge von Interaktionen gleich und nennt klassische Anwendungsfälle (engl.: use cases) als eine Form von Szenarien, was der Bedeutung eines Szenarios im Rahmen von szenariobasierten Testprozessen für automatisierte Fahrzeugsysteme nicht gerecht wird.

Trotz offensichtlicher Unterschiede weisen viele Definitionen in den wesentlichen Merkmalen aber auch eine hohe Ähnlichkeit auf. So schreiben Go & Carroll, dass die meisten Szenariobeschreibungen unabhängig von der Anwendungsdomäne, dem genutzten Medium und der konkreten inhaltlichen Form mindestens die folgenden Inhalte explizit oder implizit beschreiben: (1) Akteure, (2) Hintergrundinformationen über die Akteure und Annahmen über ihr Umfeld, (3) Ziele der Akteure und (4) Abfolgen von Handlungen und Ereignissen [Go & Carroll, 2004].

Für die Verwendung im Kontext von Tests sowie der funktionalen Beschreibung von Fahrerassistenzsystemen haben Ulbrich et al. [2015] einige der öffentlich existierenden Definitionen hervorhebend zusammengetragen und diese anschließend als Grundlage für den Vorschlag eigener konsistenter Definitionen der drei zentralen Begriffe *Szene*, *Szenario* und *Situation* verwendet. Die Konzepte hinter den Begriffen unterscheiden sich primär in ihrem Bezug zu der Zeit bzw. dem Fortschreiten der Zeit und durch die Menge der repräsentierten Informationen.

Eine *Szene* beschreibt dabei laut Ulbrich et al. alle verfügbaren Informationen über das für den Testfall relevante Umfeld, einschließlich der Szenerie, der dynamischen Elemente, der Repräsentation aller Akteure und Beobachter sowie ihrer Beziehungen zueinander zu einem festen Zeitpunkt – eine Momentaufnahme. Bei voranschreitender Zeit erhält man also für jeden Zeitpunkt t durch Aufzeichnung aller zu diesem Zeitpunkt verfügbaren Informationen eine einzigartige Szene S_t. Eine *Situation* ist eine Auswahl der Informationen, die in einer Szene enthalten sind. Eine Situation ist also eine Teilmenge einer Szene. Konkret enthält eine Situation nur diejenigen Informationen, die für einen bestimmten Akteur zum jeweiligen Zeitpunkt entscheidungsrelevant sind. Sie stellt somit die subjektive Sichtweise eines Elements auf die Szene dar. Ein *Szenario* erweitert das Konzept einer Szene um die Beschreibung der zeitlichen Entwicklung der Szenenelemente. Es setzt sich daher zusammen aus einer *Startszene* als Ausgangspunkt und der Beschreibung der Entwicklung der enthaltenen Elemente in Form von Handlungen und Ereignissen sowie den individuellen Zielen und Werten der Akteure. [Ulbrich et al., 2015] Im Gegensatz zu einer Szene, welche Informationen zu einem festen Zeitpunkt beschreibt, bezieht sich ein Szenario also auf einen Zeitraum und die Entwicklung der Szenenelemente innerhalb dieses Zeitraums. Abbildung 23 stellt die beschriebenen Zusammenhänge grafisch dar.

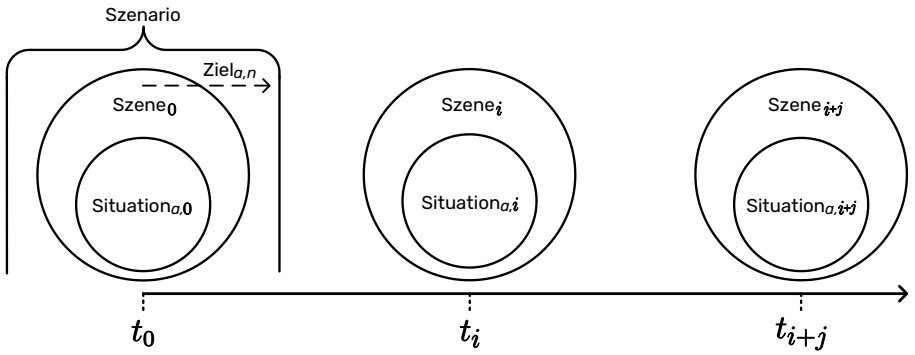


Abbildung 23: Die Beziehungen zwischen Situationen, Szenen und Szenario. Das Fortschreiten der Zeit wird von der horizontalen Achse repräsentiert, wobei t_0 der Anfangszustand ist. Die Menge aller verfügbaren Informationen über alle Akteure und ihre Umgebung zu einem bestimmten Zeitpunkt t bildet den Inhalt einer Szene. Für jeden Akteur innerhalb dieser Szene existiert zum selben Zeitpunkt t eine Situation als Teilmenge der in der Szene enthaltenen Informationen. Ein Szenario ist die Kombination einer Startszene und Zielen, Werten sowie Verhaltensbeschreibungen der Akteure.

Aufgrund ihrer Relevanz für die weiteren Inhalte dieser Arbeit, sind die zuvor in Zusammenhang gesetzten zentralen Begriffe nachfolgend zusätzlich individuell aufgelistet, angelehnt an [Ulbrich et al., 2015] definiert sowie teilweise durch Beispiele erweitert.

Akteur: Ein Akteur ist ein selbst handelndes Element. Mit Bezug zu maritimem Verkehr könnte ein Akteur je nach Abstraktionslevel der Betrachtung u. a. ein Kapitän, ein Schiff oder ein automatisiertes Steuersystem sein.

Beobachter: Ein Beobachter ist ein wahrnehmendes Element, das entweder Teil einer Szene ist oder eine Szene als Ganzes von außen betrachtet. Beispiele für Beobachter im Rahmen maritimer Testfahrten sind somit z. B. fest installierte Kamerasysteme, die lediglich zur Überwachung dienen, aber durch die Installation in unmittelbarer Nähe (z. B. auf der Kaimauer) Teil der Szene selbst sind. Die Rolle eines externen Beobachters kann u. a. einem menschlichen Prüfer im Rahmen realer Testfahrten zukommen, welcher keinen Einfluss auf die Testfälle und ihre Durchführung selbst hat, aber über Erfolg oder Misserfolg des getesteten Systems in Bezug auf die geprüfte Funktion entscheidet.

Dynamisches Element: Elemente, die sich durch vorhandene kinetische Energie oder Fähigkeiten bewegen könnten, werden als dynamisch bezeichnet. Dabei ist es laut Ulbrich et al. nicht relevant, ob das Objekt die Bewegung selbst initiiert hat oder ob die kinetische Energie extern zugeführt wurde. Sowohl Schiffe mit eigenem aktivem Antrieb als auch Treibholz sind somit als dynamisches Element zu betrachten.

Stationäres Element: Im Gegensatz zu dynamischen Elementen sind stationäre Elemente in ihrer räumlichen Position unveränderbar, können sich also nicht von selbst bewegen und im Regelfall auch nicht durch externe Einflüsse bewegt werden. Für die Unterscheidung sehen Ulbrich et al. ausschließlich den betrachteten Zeitraum als relevant an. Dadurch werden quasi-stationäre Elemente wie Wetter- und Lichtbedingungen ebenfalls den stationären Elementen zugeordnet. Stationäre Elemente können metrischer, semantischer und topologischer Natur sein.

Szene: Eine Szene bezieht sich immer auf einen eindeutigen Zeitpunkt. Sie umfasst die Menge der dynamischen Elemente mit ihren Attributen und Zuständen, die Fähigkeiten und Fertigkeiten der Akteure und Beobachter sowie die Menge aller stationären Elemente mit ihren Attributen zu diesem Zeitpunkt. Zusätzlich sind Informationen über die Beziehungen zwischen diesen Entitäten Teil einer Szene. Hervorzuheben ist, dass alle genannten Informationen statischer Natur sind – aus einer Szene kann nicht abgeleitet werden, wie die Entwicklung bei Fortschreiten der Zeit verlaufen wird.

Situation: Die Teilmenge der Informationsmenge, die durch eine Szene abgebildet wird, welche durch Auswahl derjenigen Informationen entsteht, die für einen bestimmten Akteur zum von der Szene referenzierten Zeitpunkt für sein weiteres Verhalten relevant sind, stellt eine Situation dar. Da eine Situation immer durch eine Informationsauswahl durch einen Akteur bestimmt wird, stellt sie dabei stets ein subjektives Abbild der Umgebung dar. Im Gegensatz zur Definition von Ulbrich et al. werden hier aber durch Informationsaugmentierung erzeugte Informationen nicht als Teil einer Situation betrachtet. Lediglich direkt verfügbare Informationen stellen die Situation dar, in der sich ein Akteur befindet, und eine weitere Verarbeitung dieser Informationen sowie eine Ableitung weiterer Informationen aus diesen obliegt allein der Verarbeitung durch den Akteur und sollte daher kein unmittelbarer Teil der Beschreibung sein, sondern eine ebenfalls zu testende Funktionalität.

Szenario: Aufbauend auf den zuvor eingeführten Begriffen umfasst der zentrale Begriff des Szenarios eine Szene mitsamt all ihrer Inhalte. Diese Startszene wird erweitert um eine Beschreibung der zeitlichen Entwicklung der Szenenelemente. Dafür können innerhalb eines Szenarios vorterminierte Aktionen und Ereignisse mit Bezug zu Elementen sowie Ziele und Werte definiert werden, welche bei fortschreitender Zeit Einfluss auf das Verhalten der Akteure haben. Ein Szenario führt somit mit dem Fortschreiten der Zeit zu weiteren Szenen, welche sich aus der Startszene durch Veränderungen der Elementzustände ergeben.

Wohl hauptsächlich begründet dadurch, dass die Beschreibungen der Inhalte von Szenarien, Szenen und Situationen von Ulbrich et al. trotz ihrer Bedeutung überwiegend textuell und wenig formal beschrieben werden, interpretieren aufbauende Arbeiten häufig zunächst die Beschreibungen und konkretisieren und erweitern diese teils formal. So ordnen Steimle et al. in [Steimle et al., 2021] die Begriffe für die Anwendung in Entwicklungs- und Testprozesse automatisierter Fahrzeuge ein und stellen darauf aufbauend konkrete Taxonomien vor, welche mit Hilfe von Diagrammen der grafischen Modellierungssprache *Unified Modeling Language (UML)* formal präsentiert werden. Das Diagramm zur Be-

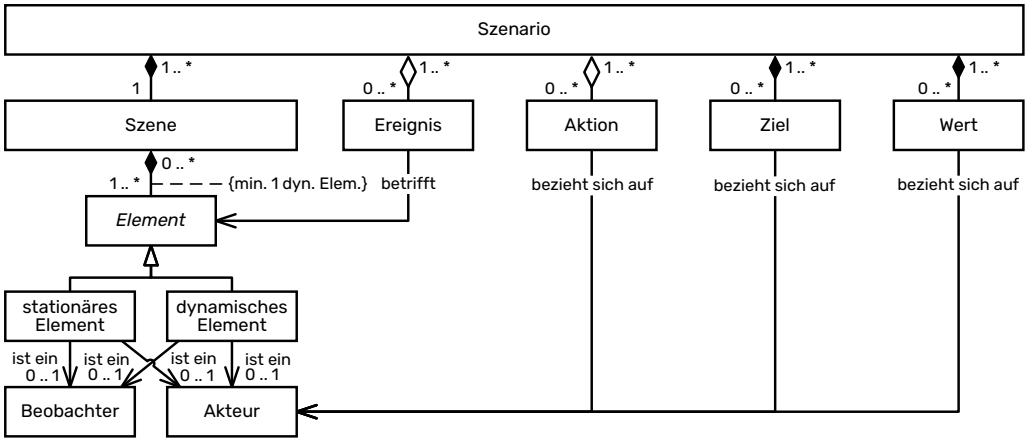


Abbildung 24: Struktur eines Szenarios und seiner Inhalte (Inhaltlich basierend auf [Ulbrich et al., 2015b]; Grafisch angelehnt an [Steimle, Menzel und Maurer, 2021])

schreibung der Taxonomie der grundlegenden Begriffe und Inhalte eines Szenarios stellt allerdings einige in dieser Arbeit als zentral betrachtete Zusammenhänge nicht eindeutig dar. In Abbildung 24 ist daher die in dieser Arbeit Anwendung findende abgeänderte Interpretation dargestellt. Die wesentlichen Unterschiede zu dem Vorschlag von Steimle et al. sind in den Beziehungen und Zusammensetzungsverhältnissen zu finden. So ist u. a. in der Arbeit von Ulbrich et al. die Aussage *„Ein Akteur ist in dieser Definition ein selbst handelndes Element. Ein Beobachter ist ein wahrnehmendes Element innerhalb der Szene oder eines, das die Szene als Ganzes betrachtet. Ein Element kann gleichzeitig Beobachter und Akteur sein […]"* zu finden, die Tatsache, dass die Rollen eines Akteurs und eines Beobachters beliebig kombiniert einem konkreten Element der Szene zugewiesen werden können, aber nicht in der vorgeschlagenen Taxonomie abgebildet. Weiter fehlt ebenso die Zuordnung von Ereignissen, Aktionen, Zielen und Werten zu den entsprechend betroffenen oder ausführenden Elementen der Szene. Hervorzuheben ist, dass in dieser Arbeit eine Abfolge von Szenen nicht als Szenario betrachtet wird und die Kombination einer einzelnen Startszene mit Akteuren und ihrem Verhalten, welche durch Ziele und Werte beeinflusst werden, ein Szenario definiert. Begründet ist diese Entscheidung zum einen dadurch, dass in dieser Arbeit der Einsatz von Simulationstechniken forciert wird und in diesem Kontext eine Abfolge von vorgegebenen Szenen, wie die namensgebende Analogie bereits aussagt, mehr einer reinen Wiedergabe historischer Daten gleichkommen würde als einer Simulation im eigentlichen Sinne. Zum anderen lässt die Definition von Ulbrich et al. bei dieser Variante offen, wie und ob das zu testende System als Akteur in den Szenen zu integrieren ist. Ein vordefiniertes Verhalten wäre im Rahmen eines Testlaufs nicht zielführend, da in diesem Fall das dynamische Verhalten dieses Elements beobachtet und ausgewertet werden muss. Eine Abfolge von Szenen wird stattdessen als die mögliche Ausgabe einer Verkehrssimulation angesehen (vgl. Definition einer *Verkehrstrajektorie* in Abschnitt 7.3.3)

7.3 COMPUTERGESTÜTZTE TESTDURCHFÜHRUNG

Der Einsatz von Computersimulationen bei der Durchführung szenariobasierter Testreihen zur Verifikation und Validierung automatisierter und autonomer Fahrzeuge erweist sich aus mehreren Gründen als sinnvoll und hat sich mittlerweile vor allem für Straßenfahrzeuge als weitverbreitete Praxis etabliert (vgl. u. a. [Nalic et al., 2021; Alghodhaifi & Lakshmanan, 2021; Riedmaier et al., 2020; Jesenski et al., 2019]). Mithilfe von Computersimulationen ist es möglich, komplexe und kritische Szenarien

systematisch und wiederholt durchzuführen, die in der Realität entweder nur unter großem Zeitaufwand, riskant oder gar nicht reproduzierbar wären (vgl. Abschnitt 2). Dies ist insbesondere im Bereich autonomer Schiffe von entscheidender Bedeutung, da die maritimen Umgebungen häufig von extremen Wetterlagen, eingeschränkten Sichtverhältnissen sowie vielfältigen Verkehrs- und Navigationsherausforderungen geprägt sind.

Des Weiteren ermöglichen Simulationen eine umfassende Abdeckung von seltenen, häufig an den Rändern des abzudeckenden Parameterraums befindlichen, aber sicherheitsrelevanten Bedingungen und Ereignissen, die in physischen Tests nicht ohne Gefährdung von Personal, Material oder Umwelt abgebildet werden können. Der Einsatz durch Computer simulierter Umgebungen und Verkehrsteilnehmer ermöglicht es, automatisierte Schiffe und deren Teilsysteme bereits in frühen Entwicklungsphasen zu prüfen. Ein so mögliches frühes Finden von Fehlern führt in der Regel zu einer Beschleunigung der Entwicklungsdauer, einer Reduktion der Kosten und einer Minimierung der Risiken (vgl. [Karthaus et al., 2021; Dierßen, 2002]). Zusätzlich ermöglichen Computersimulationen, Tests zu vereinheitlichen und reproduzierbar zu gestalten, wodurch die Vergleichbarkeit der Testergebnisse verbessert wird. Die Erfüllung regulatorischer Anforderungen sowie die Vorbereitung des sicheren Einsatzes automatisierter Schiffe im realen Betrieb sind maßgeblich von der Standardisierung der zur Absicherung eingesetzten Prozesse und Werkzeuge abhängig. Computersimulationen sind folglich ein essenzielles Werkzeug, um die Zuverlässigkeit, Sicherheit und Robustheit maritimer automatisierter Fahrsysteme umfassend sicherzustellen und eine weitverbreitete Einführung dieser zu ermöglichen. Aufgrund der hohen Relevanz von Computersimulationen für diese Arbeit – bereits der Titel enthält die Worte *Modellierung und Simulation* – werden in diesem Abschnitt relevante Konzepte aus diesem Themengebiet näher betrachtet, definiert und in Verbindung zueinander gesetzt. Dabei wird der anfangs noch breite Fokus kontinuierlich auf die Anwendung zur Unterstützung szenariobasierter Validierung und Verifizierung geschärft.

7.3.1 SIMULATION

Der Duden[23] definiert *simulieren* unter anderem als die Tätigkeit, „*Sachverhalte [und] Vorgänge mit technischen [oder] naturwissenschaftlichen Mitteln modellhaft zu Übungs-[oder] Erkenntniszwecken nach[zu]bilden*“. Anhand dieser Definition ist zu erkennen, dass, obwohl heutzutage zumeist mit einer Computersoftware zur Berechnung und Darstellung von Zusammenhängen und Verläufen gleichgesetzt, der Begriff *Simulation* jegliche Nachbildung von existierenden oder geplanten Dingen und Prozessen sowie das Experimentieren mit diesen umfasst. In der Aerodynamik beispielsweise wurde das Strömungsverhalten von Tragflächen und Flugzeugkonstruktionen traditionell in Windkanalexperimenten untersucht und Astronauten trainieren in simulierter Schwerelosigkeit, indem sie sich unter Wasser in speziellen Anzügen befinden [Dierßen, 2002].

Essenzielles Kriterium, um als Simulation zu gelten, ist dabei die Übertragbarkeit der Ergebnisse in die Realität. Das verdeutlicht auch die Definition, die der Verein Deutscher Ingenieure (VDI) in seiner Richtlinie 3633 anführt: „*Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind; insbesondere werden die Prozesse über die Zeit entwickelt.*“ [VDI, 3363] Ein weiteres Merkmal einer Simulation ist, dass die Betrachtung der nachgebildeten Entitäten auf das Verhalten und die Veränderungen

[23] `https://www.duden.de/rechtschreibung/simulieren#Bedeutung-1` [letzter Zugriff: 17.02.26]

dieser im Laufe der Zeit ausgerichtet ist [Serena et al., 2023]. Ohne die Zeitkomponente wäre die Nachbildung im Bereich der statischen *Repräsentationen* zu verorten. Zu diesem Zweck werden mathematische oder logische Modelle verwendet, die das Verhalten der beteiligten Einheiten und ihre Interaktionen beschreiben (Modelle und ihre tragende Rolle in Bezug auf Simulationen werden in den Abschnitten 7.3.1 und 7.4 im Detail betrachtet).

Simulationen jeder Art sind Hilfsmittel für den Umgang mit der Realität [Bossel, 2004]. Früher hat man sich vor der Umsetzung eines größeren Vorhabens ausführlich Gedanken gemacht, umfangreiche Berechnungen manuell durchgeführt und handschriftliche Pläne erstellt. Die erstellten Konzepte wurden anschließend häufig anhand von Nachbildungen kleineren Maßstabs bezüglich ihrer Eignung getestet. Heutzutage greift man dabei oftmals auf computergestützte Verfahren zurück, da diese eine Reihe von Vorteilen bieten, wie die deutlich zeiteffizientere Durchführung von Berechnungen und die zuverlässige Abbildung von komplexen Zusammenhängen. Das Digitale Wörterbuch der deutschen Sprache (DWDS)[24] definiert eine Computersimulation als *„modellhafte Nachbildung von komplexen Vorgängen, Prozessen oder Zuständen mithilfe von Computern bzw. Computerprogrammen*". Die Ursprünge der Computersimulation können auf die Pionierarbeit von Ulam und von Neumann zurückverfolgt werden, die im Rahmen von Berechnungsexperimenten zur Neutronenstreuung die Monte-Carlo-Methode formulierten [Zeigler et al., 2023; Goldsman et al., 2010].

Typische Ziele, die durch den Einsatz von Computersimulationen im Laufe der letzten Jahrzehnte zu erreichen versucht wurden, sind die Optimierung von Systemverhalten, das Anbieten von Entscheidungshilfen, die Erstellung verlässlicher Prognosen von Systemtrajektorien, die Überprüfung eines Systementwurfs auf Eignung, die Überprüfung von Theorien durch hypothetische Modellbildung sowie das gezielte Üben bestimmter Situationen im Ausbildungsbereich [Mattern, 1996].

Durch die stete Leistungssteigerung von Prozessoren sind wir heute an einem Punkt, an dem auch kaum durch den Menschen in Gänze erfassbare Zusammenhänge und Entwicklungen durch Computer dargestellt, simuliert und analysiert werden können. Obwohl Simulationsergebnissen und Studien, die auf Computersimulationen basieren, häufig ein gewisses Mistrauen entgegengebracht wird [Uhrmacher et al., 2016; Merali, 2010; Kurkowski et al., 2005], hat sich der Einsatz von Simulationen in verschiedensten Bereichen bewährt. Dieses Misstrauen ist hauptsächlich durch zwei Dinge verursacht: Simulationen, Tools, Modelle etc., die nicht öffentlich zugänglich sind und somit die Studienergebnisse/Simulationsergebnisse nicht reproduzierbar sind (Prüfbarkeit), und die ablehnende Tendenz vieler Anwender stochastischen Verteilungen gegenüber, welche häufig in Simulationen anstelle von echten Daten und Statistiken eingesetzt werden [Uhrmacher et al., 2016]. So wurden schrittweise fast alle analogen Darstellungsmöglichkeiten, wie hydraulische, elektrische und mechanische Modelle, in der Entwicklung neuer Systeme durch Computersimulationen abgelöst. Die Gründe dafür sind offensichtlich: (1) Verschiedenste Systeme können mit einer einheitlichen Methodologie untersucht werden, (2) die Kosten sind in den meisten Fällen deutlich niedriger, verglichen mit einer Untersuchung anhand physikalischer Modelle, (3) der zeitliche Ablauf langwieriger Prozesse kann beschleunigt betrachtet werden und schnelle Abläufe zur Verbesserung der Beobachtbarkeit verlangsamt, (4) Bedingungen und Abläufe, die für das System und seine Umgebung potenziell gefährlich sind, können sicher und frei

[24] `https://www.dwds.de/wb/Computersimulation` [letzter Zugriff: 17.02.26]

von Konsequenzen untersucht werden, (5) Eingriffe und Veränderungen am realen System sind zur Untersuchung der Auswirkung dieser nicht notwendig. [Bossel, 2004]

Die nachfolgende Liste enthält einige illustrative Beispiele für Werkzeuge im Bereich der Computersimulationen, die ihren Anwendungsdomänen entsprechend gruppiert sind. Die Vielfältigkeit der Listeninhalte lässt die große Bandbreite des Einsatzes von Computersimulationen erahnen.

↳ **Straßenverkehr**
- TORCS[25]: The Open Racing Car Simulator [Wymann et al., 2016]
- CARLA[26]: Open-Source Simulationsplattform zur Forschung zum autonomen Fahren [Dosovitskiy et al., 2017]
- Einsatz der kommerziellen Simulationssoftware CarMaker[27] zur Evaluation eines Modells zur dynamischen Geschwindigkeitsbegrenzung [Graubohm et al., 2023]
- Risikobewertung für autonome Shuttles [Rehrl et al., 2020]

↳ **Luftfahrt**
- AirTrafficSim[28]: Offene webbasierte Plattform zur Simulation von Luftverkehr [Hui et al., 2023]
- AirSim[29]: Physikalische hochauflösende Simulation für Entwicklung, Training und Bewertung von Algorithmen zur Steuerung von automatisierten Luftfahrzeugen. [Shah et al., 2018]

↳ **Raumfahrt**
- Verteilte Simulation von Raumfahrtmissionen [Argüello & Miró, 2000]

↳ **Maritime Operationen und maritimer Verkehr**
- FhSim[30]: Software-Plattform für mathematische Modellierung und numerische Simulation mit Fokus auf maritime Anwendungen industriellen Fischfangs [Reite et al., 2014]
- Maritime Traffic Simulation (MTS)[31]: Simulation von maritimem Verkehr zur Generierung von Testfällen, Erprobung von Verkehrsmanagementkonzepten sowie der Validierung anderer maritimer Anwendungen. [Rüssmeier et al., 2019; Dibbern & Hahn, 2014]
- Simulation logistischer Abläufe eines Industriehafens zu Trainingszwecken [Massei et al., 2013]

↳ **Menschliches Verhalten**
- Simulation des Verhaltens von Menschen in Warteschlangen und der Auswirkung von Zeitpunkten zur Mitteilung der geschätzten restlichen Wartezeit [Dai et al., 2024]
- Simulation von Fußgängern im Stadtzentrum von Posen im Verlauf eines Tages [Wozniak, 2024]

↳ **Elementarrisiken, Umweltkatastrophen, Krisen- und Rettungsszenarien**
- Überblick über verteilte Plattformen zur Simulation natürlicher Gefahren und der Erprobung von Notfallplänen [Xu et al., 2021]
- Simulation der Auswirkungen von Erdbebenkatastrophen unter Einbeziehung des menschlichen Verhaltens am Beispiel der Stadt Beirut [Iskandar et al., 2024]
- Simulation von Evakuierungsverläufen bei Bränden unter Berücksichtigung von Leitfaktoren am Beispiel des U-Bahn-Umsteigebahnhofs der Stadt Shenyang [Liu et al., 2024b]

[25] https://sourceforge.net/projects/torcs/ [letzter Zugriff: 17.02.26]
[26] https://carla.org/ [letzter Zugriff: 17.02.26]
[27] https://www.ipg-automotive.com/de/produkte-loesungen/software/carmaker/ [letzter Zugriff: 17.02.26]
[28] https://github.com/HKUST-OCTAD-LAB/AirTrafficSim [letzter Zugriff: 17.02.26]
[29] https://microsoft.github.io/AirSim/ [letzter Zugriff: 17.02.26]
[30] https://fhsim.smd.sintef.no/ [letzter Zugriff: 17.02.26]
[31] https://www.dlr.de/de/se/forschung-transfer/forschungsinfrastruktur/emir [letzter Zugriff: 17.02.26]

- Simulationen von Epidemieverläufen zum Vergleich von Krisenbewältigungsstrategien [Adam & Arduin, 2024]

↳ **Training künstlicher Intelligenz**

- AI2-THOR[32]: Framework zur Simulation interaktiver Umgebungen für das Training physischer Agenten verkörperter künstlicher Intelligenz [Kolve et al., 2022]
- Gibson Env[33]: Training und Erprobung von Wahrnehmung und Navigation aktiver Agenten in virtualisierten Varianten echter Gebäude [Xia et al., 2018]
- Habitat[34]: Plattform für das Training physischer Agenten verkörpert künstlicher in virtuellen fotorealistischen Umgebungen [Savva et al., 2019]

↳ **Multi Robot Simulation**

- Gazebo[35]: Simulierte Multi-Agenten Umgebung für Entwicklung, Training und Erprobung von interagierenden Robotersystemen [Koenig & Howard, 2004]
- Urban Search And Rescue Simulation (USARSim)[36]: Multi-Agenten Umgebung für Forschung und Bildung, die als Basis für die Robocup Rescue League eingesetzt wird [Carpin et al., 2007]

↳ **Wirtschaftliche Prozesse und Systeme**

- Simulation von Strommärkten mit konkurrierenden Speichereinheiten [Harder et al., 2023]
- Framework für die Simulation der Interaktionen zwischen Großhändlern, Geschäften und Kunden im Lebensmitteleinzelhandel mit dem Ziel, künstliche Intelligenzen zu trainieren, die zur Verringerung der Lebensmittelverschwendung beitragen. [Pilarski et al., 2025]
- Einsatz von Simulation zur Verbesserung des Stahlherstellungsprozesses unter Berücksichtigung des Transports [Tseng & Aldi Winata, 2025]

↳ **System- und Produktentwicklung**

- Komponentenorientierte Simulation von Maschinen und Anlagen [Dierßen, 2002]
- Einsatz von Simulationen zur Ermöglichung von kontinuierlicher Integration in der Automobilindustrie [Knauss et al., 2016]

↳ **Netzwerke**

- Simulation von Stromnetzen mit Fokus auf die Stromversorgung von Wohngebieten zur Untersuchung des Zusammenspiels von Elektrofahrzeugen, Hausspeichersystemen und Fotovoltaik-Anlagen auf der Niederspannungsebene [Nobis, 2016]
- Simulation der Datenübertragung innerhalb eines Satellitennetzwerks zur Untersuchung auftretender Kommunikationsverzögerungen [Moreira et al., 2025]

↳ **Astronomie**

- Simulation der Bewegung von galaktischen Objekten [Kowalk, 2003]

Begriffsklärung: Simulation

Die Begriffe *Simulation* und *Computersimulation* werden häufig analog zueinander genutzt, obwohl *Computersimulation* lediglich eine Teilmenge der Konzepte, die der Menge *Simulationen* angehören, beschreibt (vgl. vorangegangene Einleitung zum Abschnitt 7.3.1). So ist in dem vom Modeling and Simulation Coordination Office des US-amerikanischen Department of Defense veröffentlichten

[32] `https://ai2thor.allenai.org/` [letzter Zugriff: 17.02.26]
[33] `http://gibsonenv.stanford.edu/` [letzter Zugriff: 17.02.26]
[34] `https://aihabitat.org/` [letzter Zugriff: 17.02.26]
[35] `https://gazebosim.org/` [letzter Zugriff: 17.02.26]
[36] `https://sourceforge.net/projects/usarsim/` [letzter Zugriff: 17.02.26]

Glossar *Simulation* definiert als eine Methode zur Implementierung eines Modells mit Bezug zur fortschreitenden Zeit [Department of Defense, Modeling and Simulation Coordination Office, 2011]. Zusätzlich wird eine *Simulationsanwendung* (engl.: simulation application) explizit beschrieben als eine Software, die auf einem Computer ausgeführt wird und echtweltliche Phänomene repräsentiert bzw. simuliert [Department of Defense, Modeling and Simulation Coordination Office, 2011]. In dieser Arbeit ist nachfolgend aus Gründen der Effizienz und Lesbarkeit mit dem Begriff *Simulation* in den meisten Fällen eine Computersimulation gemeint.

Wie auch den beiden angeführten Definitionen entnommen werden kann, ist der Begriff der Modellierung fest mit dem der Simulation verknüpft. Die entsprechende Fachdisziplin wird im Allgemeinen ebenfalls zusammenfassend als *Modellbildung und Simulation* (engl.: Modelling and Simulation) *(M&S)* bezeichnet [Ali et al., 2024; Fujimoto & Loper, 2017; Bungartz, 2013; Bossel, 2004]. Die funktionalen Abläufe einer Simulationsanwendung können daher nicht vollständig dargestellt werden, ohne auf das Konzept eines Modells und die Tätigkeit der Modellierung zu verweisen, mit welchen sich der nächste Unterabschnitt beschäftigt.

Für diese Arbeit wurde folgende Definition formuliert und diese entsprechend im weiteren Verlauf als gültig betrachtet: Eine Simulation bzw. Simulationsanwendung ist eine Software, die in der Lage ist, Modelle realer Systeme auszuführen. Wobei *Ausführen* hier die Weiterrechnung des Systemzustands bei Fortschreiten der Zeit bedeutet (vgl. Abbildung 25). Eine Simulation alleine ist somit streng genommen kein vollständiges und direkt ausführbares Anwendungsprogramm [Gomes et al., 2019; Dierßen, 2002], sondern benötigt zusätzlich noch Eingaben in Form von Modellen und Parametrierungen dieser, um ihren Zweck zu erfüllen.

Begriffsklärung: Modell

Simulationen, unabhängig davon, welchem konkreten Zweck sie dienen, bilden immer ein oder mehrere reale Systeme ab. Im Kontext von Simulation wird dabei von dem *zu simulierenden* und dem *simulierten System* gesprochen. Das zu simulierende System kann ein reales, bereits existierendes System sein, ein sich in der Entwicklung befindliches System oder auch ein bisher lediglich geplantes.

Systeme sind im Allgemeinen komplexe Gebilde und ihre *vollständige* Beschreibung ist schwierig bis unmöglich. Daher werden Systeme für wissenschaftliche Untersuchungen zumeist auf Modelle reduziert, welche die relevanten Merkmale abbilden. Ein Beispiel dafür sind Atommodelle, die die anschauliche Erklärung von vielen physikalischen Phänomenen erlauben und im Laufe der Jahrhunderte immer weiter verfeinert wurden. [Kowalk, 1996]

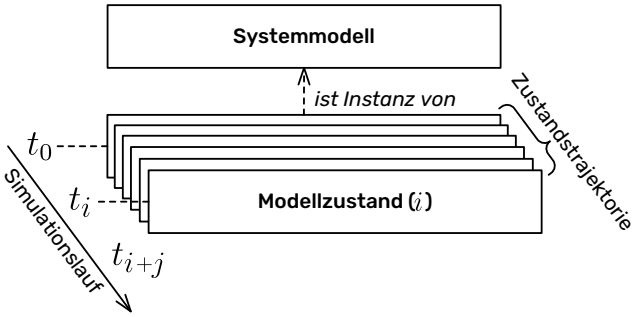


Abbildung 25: Zusammenhang der Begriffe (System-)Modell, (Modell-)Zustand, Simulationslauf und Zustandstrajektorie im Kontext einer zeitbasierten Simulation

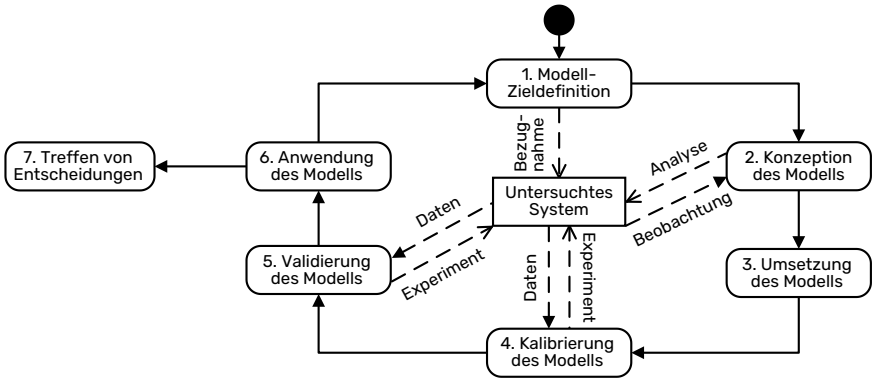


Abbildung 26: Vereinfachtes Model des *Modelling & Simulation* Prozesses (angelehnt an [Ali et al., 2024])

Modelle und Simulationen sind somit Hilfsmittel für den Umgang mit der Realität: Menschen haben schon immer gedanklich Pläne erstellt und diese anschließend ebenfalls in Gedanken durchgespielt, die Resultate den Mitmenschen mitgeteilt und die Pläne anschließend angepasst, umgesetzt oder verworfen. Ein solcher gedanklicher Plan einer realen Abfolge von Aktionen ist ein Modell eben jener Abfolge. Die Spannweite von Modellen geht dabei aber weit über gedankliche Konstrukte hinaus und reicht von verkleinerten Versionen des Originals über Schnittzeichnungen, Ablaufpläne, Funktionsdiagramme und mathematische Formeln bis zu Computerprogrammen. [Bossel, 2004]

Das zu simulierende System wird durch ein entsprechendes *Systemmodell* repräsentiert. Unter einem Modell versteht man allgemein ein vereinfachendes Abbild einer partiellen Realität [Bungartz, 2013]. Modelle sind also immer Abstraktionen der Realität und bilden diese nicht – zumindest nicht in allen Bereichen – exakt nach, sondern bilden Eigenschaften des Systems zusammenfassend nach [Kowalk, 1996]. Dabei wird immer vereinfacht, zusammengefasst und/oder weggelassen, also eine Auswahl getroffen und Entscheidungsvorgänge durchgeführt [Bossel, 2004]. Wird das resultierende Modell mithilfe der Simulationsanwendung und einer spezifischen Parametrierung ausgeführt, dann handelt es sich um einen *Simulationslauf*. Aus einem Simulationslauf ergibt sich eine Zustandstrajektorie (vgl. Abschnitt 6.3) für das simulierte System. Die gewonnene Zustandstrajektorie kann anschließend ausgewertet werden und die so gewonnenen Erkenntnisse können auf das reale System übertragen werden. Die Übertragbarkeit ist dabei abhängig von der Güte der Repräsentation des realen Systems durch das abstrakte Modell, welche wiederum anwendungsfallspezifisch ist.

Eine Simulation ist dabei kein monolithischer Prozess, sondern setzt sich aus einer Abfolge von mehreren Teilschritten zusammen [Bungartz, 2013]. Üblicherweise werden dabei die Schritte mehrfach durchlaufen, da der Gesamtprozess selbst eine Art Feedbackschleife bildet. Abbildung 26 zeigt eine Variante des Modellierungs- und Simulationsprozesses, welche sich aus sieben Teilschritten zusammensetzt. Den Anfang macht dabei immer eine genaue Definition des Ziels, das durch den Einsatz des Modells erreicht werden soll, denn ein Modell wird immer auf die Erfüllung eines bestimmten Zwecks ausgerichtet erstellt, dem *Modellzweck*. Neben den üblichen Gründen einer zielgerichteten Konzeption und Umsetzung, wie man sie auch aus der allgemeinen Softwareentwicklung kennt, ist die strenge Beschränkung des Modellzwecks auch eine Frage der Effizienz und Anwendbarkeit. Denn der aus der Aufgabenstellung (dem zu erreichenden Ziel) abgeleitete Modellzweck beeinflusst wiederum den Umfang der Modellformulierung (dem eigentlichen Modell selbst) [Bossel, 2004]. Bossel formuliert die Notwendigkeit eines streng eingeschränkten Modellzwecks drastischer: Ein allgemeingültiges Supermodell ist nur mit hohem Aufwand erstellbar und wäre für spezielle Problemstellungen ineffizient.

Weil mit der Komplexität auch die Fehlermöglichkeiten anwachsen, ist zu erwarten, dass für spezielle Fragen die Zuverlässigkeit und Aussagekraft gering sind. Generell gilt also nicht, dass das größere Modell auch das bessere ist:

> *„Das beste Modell ist dasjenige, das seinen Zweck bei geringstmöglicher Komplexität voll erfüllt. Das Modell sollte so einfach wie möglich, aber so komplex wie nötig sein."*
> [Bossel, 2004]

Daraus folgt schlussendlich, dass ein System für unterschiedliche Untersuchungsziele durch unterschiedliche Modelle abgebildet werden muss. Bungartz argumentiert ähnlich, wenn auch weniger drastisch, und beschreibt Modellierung als eine Frage der Abwägung von Aufwand und Genauigkeit. Je mehr Details und Einzeleffekte integriert werden, desto präziser werden die Simulationsergebnisse, zum Preis steigender Modell-Entwicklungs- und Simulationskosten [Bungartz, 2013]. Eine allgemeine Definition, wann ein Modell korrekt ist, wie die in der Informatik übliche Formulierung, Korrektheit bedeute, dass sich ein Modell eines Systems genauso verhält wie das reale System, muss aus den genannten Gründen ebenfalls genauer spezifiziert werden. Kowalk schlägt unter anderem für diesen Zweck vor, die Korrektheit dann anzunehmen, wenn die Modellkorrektheit durch eine gewisse Anzahl von Experimenten nicht widerlegt werden konnte. Darauf aufbauend soll Modellkorrektheit im Rahmen dieser Arbeit dann angenommen werden, wenn die durchgeführten Experimente dem durch den zuvor definierten Modellzweck begrenzten Umfang entsprechen.

Begriffsklärung: Zeit

Das Konzept eines Modells ist definitionsgemäß eine statische Darstellung eines Ausschnitts der realen Welt. Erst die Simulationsdurchführung in Form eines Simulationslaufs fügt einem Modell einen zeitlichen Aspekt hinzu. Das geschieht durch die Darstellung davon, wie sich das modellierte System im Laufe der Zeit verändert. Simulation kann also als eine zeitlich veränderliche Darstellung eines Modells definiert werden. [Sokolowski & Banks, 2009]

In Bezug auf Simulationen werden drei gängige Arten von *Zeit* unterschieden: die *Simulationszeit* (engl.: simulation time), die *Echtzeit* (engl.: real time) und die *verstrichene Echtzeit* (engl.: wall-clock time). Die Simulationszeit kann dabei mit der Echtzeit und der verstrichenen Echtzeit übereinstimmen. Ist das Fortschreiten der Simulationszeit synchron zum Fortschreiten der Echtzeit, wird eine realistische Darstellung der menschlichen Wahrnehmung ermöglicht. Überdies besteht aber die, zuvor bereits als Vorteil des Einsatzes von Computersimulationen angeführte, Möglichkeit, das Fortschreiten der Simulationszeit zu beschleunigen oder zu verlangsamen. Des Weiteren werden Simulationen anhand dessen unterschieden, wie und wann die Zeit fortschreitet. Da dies eines der wesentlichen Merkmale zur Unterscheidung verschiedener Simulationstypen ist und eine erhebliche Auswirkung auf die Anwendbarkeit der jeweiligen Simulationsanwendung auf bestimmte Untersuchungsgegenstände hat, wird das Thema der Kontinuität in Abschnitt 7.3.2 separat behandelt.

Begriffsklärung: Verhalten

Der Zustand eines Systems kann mit Bezug zur Zeit statischer oder dynamischer Natur sein. *Statisch* bedeutet in diesem Fall, dass das System seinen Zustand auch bei Fortschreiten der Zeit nicht verändert. *Dynamische* Systeme hingegen ändern ihren Zustand im Laufe der Zeit [Bossel, 2004; Kowalk, 1996]. Demnach ist eigentlich jedes System dynamisch, auch wenn es kurzfristig von außen betrachtet

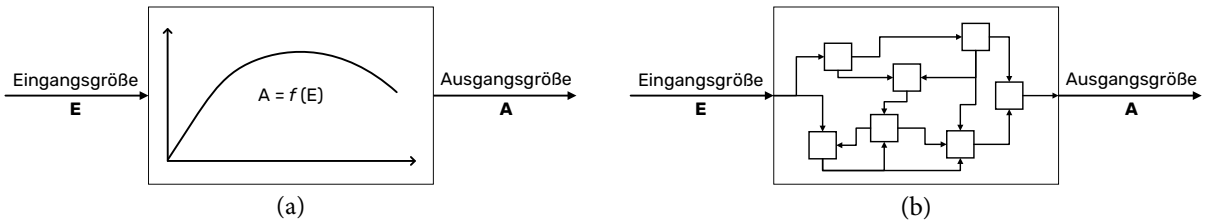
nicht so erscheinen mag, da auch diese zumeist über einen sehr langen Zeitraum betrachtet z. B. Veränderungen in Form von Alterserscheinungen durch dynamische Belastungen aufweisen [Bossel, 2004]. Da die Modellierung eines realen Systems immer mit Vereinfachung einhergeht, werden hier lediglich Systeme als dynamisch betrachtet, die in dem vom Modellzweck bestimmten Zeitraum und Abstraktionsgrad ihren Zustand ändern. Eine solche dynamische Veränderung des Systemzustands wird auch als das *Verhalten* des Systems bezeichnet [Bossel, 2004].

Gegenstand simulativer Untersuchungen ist immer das Verhalten eines Systems bzw. stellvertretend des Systemmodells. Ein zur Verhaltenssimulation geeignetes Modell muss dabei in der Lage sein, dynamisches Verhalten zu generieren. Dazu ist es erforderlich, dass es auf Einflüsse seiner simulierten Umgebung mit Veränderung seiner Zustandsgrößen reagieren kann [Bossel, 2004]. Zur Nachbildung des dynamischen Verhaltens eines realen Systems existieren zwei Herangehensweisen:

(1) Nachahmung des Verhaltens des realen Systems durch ein beliebiges Modellsystem, das lediglich der Anforderung genügen muss, gleiches Verhalten zu zeigen [Bossel, 2004]. Das Modell soll in diesem Fall lediglich bei gleichen anliegenden Eingangsgrößen die gleichen Ausgangsgrößen liefern wie das reale System. Das System wird in diesem Fall wie eine *Blackbox* behandelt, innere verhaltensbestimmende Einzelheiten und Funktionen also nicht ermittelt und stattdessen meist auf Basis historischer Daten des realen Systems die Zusammenhänge der Eingangs- und Ausgangsgrößen durch mathematische Funktionen abgebildet. Das resultierende Modell ist ein *beschreibendes* Verhaltensmodell [Bossel, 2004] (vgl. Abbildung 27a).

(2) Nachbildung der Wirkungsstrukturen des realen Systems [Bossel, 2004]: Im Gegensatz zu (1) wird hier ein Modell des Systems, nicht ein Modell des resultierenden Verhaltens allein, entwickelt. Das bedeutet, dass die Systemstruktur und die Systemprozesse des Originalsystems identifiziert, verstanden und nachgebildet werden müssen. Aufgezeichnete Daten zum Zusammenhang von Ein- und Ausgangsgrößen des Realsystems sind hingegen nicht nötig. Dieser Ansatz ist häufig arbeitsintensiv, da viel Wissen notwendig ist: Aus welchen Teilen besteht das System? Wie sind diese Teile miteinander verknüpft? Wie beeinflussen sich die Teile gegenseitig? (vgl. dazu auch Abschnitt 6.3) Das reale System muss somit im Sinne einer *Whitebox* einsehbar sein. Das resultierende Modell ist ein *erklärendes* Verhaltensmodell [Bossel, 2004] (vgl. Abbildung 27b). Im Rahmen der technischen Systementwicklung wird dieser Ansatz traditionell überwiegend eingesetzt, da technische Systeme im Allgemeinen bzgl. ihrer Wirkstruktur präzise definiert sind und sich die Wirkungszusammenhänge gut durch informatische Modelle beschreiben lassen [Bossel, 2004].

Zusätzlich zu den beiden genannten Reinformen der Verhaltensmodellierung ist auch eine Mischform möglich. Üblicherweise sind dabei die wichtigsten inneren Strukturen des realen Systems bekannt und entsprechend erklärend im Modell abgebildet. Einzelne Teile können aber durch beschreibende Mo-



(a) (b)

Abbildung 27: Modellierung von dynamischem Verhalten: (a) beschreibendes Verhaltensmodell, (b) erklärendes Verhaltensmodell (angelehnt an [Bossel, 2004])

delle wie mathematische Funktionen abgebildet sein. Gründe für eine solche Mischform können u. a. fehlende Einsehbarkeit einzelner Systemteile oder ein bewusst hoher Abstraktionsgrad sein.

Je nach gewähltem Vorgehen sind für die Konzeption und Implementierung des Modells andere Teilschritte nötig, weshalb die Entscheidung im Voraus zusammen mit dem Modellziel definiert werden sollte. So umfasst Schritt 2 des in Abbildung 26 gezeigten Prozesses generell die Teilschritte (1) *Identifizierung und Definition der Systemgrenzen*, (2) *Analyse und Konzeption der Verhaltensabbildung* und (3) *Konzeptionelle Entwicklung des Simulationsmodells*. Soll dynamisches Verhalten in das Modell mittels eines erklärenden Verhaltensmodells integriert werden, so unterteilt sich Teilschritt 2 weiter in (2.1) *Quantitative Untersuchung der Wirkungsstruktur*, (2.2) *Qualitative Untersuchung der Wirkungsstruktur*, (2.3) *Dimensionale und funktionale Untersuchung der Wirkungsstruktur* und (2.4) *Quantifizierung der Wirkungsstruktur* [Bossel, 2004].

Begriffsklärung: Prozess

Ein Prozess kann aus der systemtheoretischen Sicht definiert werden als jede Form einer zeitlichen und/oder räumlichen Veränderung. Innerhalb eines solchen Prozesses können eine oder mehrere Größen zeit- und/oder ortsabhängig variieren. Dabei werden Variablen üblicherweise entsprechend ihrer zeitlichen und/oder räumlichen Variabilität unterschieden: Variablen, die von der Zeit und/oder dem Ort abhängen, werden als Prozessvariablen bezeichnet. Demgegenüber werden Variablen, die konstant bleiben oder sich nur geringfügig ändern, als Systemparameter bezeichnet. [Dierßen, 2002; Kramer & Neculau, 1998] (vgl. Abschnitt 6.3 „Systembegriff“). Beide Varianten gilt es im festgelegten Maß (vgl. Definition des Modellzwecks im vorigen Abschnitt) während der Systemanalyse zu identifizieren und im Rahmen der Implementierung im Modell zu integrieren. Prozessvariablen sind dabei essenzielle Bestandteile des Verhaltens oder werden durch das Verhalten bestimmt. Das Verhalten eines Systems ist somit durch die Gesamtheit aller Systemprozesse geprägt. Diese Prozesse können dabei unmittelbar von außen beobachtbare Effekte haben – und somit ein für den Beobachter offensichtlicher Bestandteil des Verhaltens sein – oder sich lediglich innerhalb des Systems auswirken. Diese internen Auswirkungen können dann wiederum Auswirkungen auf den Verlauf anderer Prozesse haben. Solche durch indirekte Kaskadierung für den Systemuntersuchenden nicht sofort offensichtlichen Prozesse sind daher in den meisten Fällen jedoch essenziell für eine akkurate Nachbildung des Verhaltens (vgl. Abbildung 27b).

7.3.2 KLASSIFIKATION

Im vorangegangenen Abschnitt wurden bereits einige Gestaltungsarten von Simulationsmodellen identifiziert, welche sich in der Regel gegenseitig ausschließen. So sind ein Modell und die Simulation dieses Modells entweder statisch oder dynamisch. Auch wurden zwei Herangehensweisen zur Modellierung dynamischer Systeme vorgestellt: verhaltensbeschreibend und verhaltenserklärend. Überdies existiert eine Vielzahl von weiteren Ausprägungen, anhand welcher Systeme und Modelle dieser weiter unterschieden werden, von denen die relevantesten nachfolgend paarweise aufgelistet und erklärt werden [Bossel, 2004].

deterministisch ↔ stochastisch – Ein deterministisches Simulationsmodell führt bei gleicher Parameterwahl immer zur Erzeugung der gleichen Zustandstrajektorie. Dadurch ist bei ausschließlicher Verwendung deterministischer Modelle ein einzelner Simulationslauf ausreichend, um ein verlässliches Untersuchungsergebnis zu erzielen. [Dählmann, 2022]

In stochastischen Modellen werden hingegen zufallsbasierte Einflüsse integriert [Serena et al., 2023]. Solche Einflüsse können etwa zufällige Veränderungen von Parametern oder Wirkbeziehungen, aber auch direkte Beeinflussungen von Verhaltensabläufen, Eingangs- oder Störgrößen basierend auf Wahrscheinlichkeitsverteilungen sein. Obwohl die einzelnen Simulationsläufe so zu unterschiedlichen Ergebnissen führen, kann durch eine große Zahl durchgeführter Simulationsläufe eine Prognose über das wahrscheinlichste Verhalten und die zu erwartende Streuung getroffen werden [Dählmann, 2022; Bossel, 2004].

konstante Parameter ↔ zeitvariante Parameter: Die Parameter eines Systemmodells haben u. a. Einfluss auf die Wirkbeziehungen der Systemkomponenten. Konstante Parameter erfahren während eines Simulationslaufs keine Änderung. Zeitvariante Parameter hingegen stellen Funktionen der Simulationszeit dar, was mit dem Fortschreiten dieser zu einer Veränderung des Systemverhaltens führt. [Bossel, 2004] Ein greifbares Beispiel für ein solches zeitvariantes Verhalten sind Fahrzeugführer: Ist ein Mensch eine lange Zeit wach, so wird dieser unabhängig von der Tageszeit müde, seine Aufmerksamkeit, Konzentrationsfähigkeit und Reaktionsfähigkeit werden geringer. Das Verhalten im Straßenverkehr insgesamt verändert sich somit abhängig von der Zeit, die dieser Fahrzeugführer wach ist.

zeitdiskret ↔ zeitkontinuierlich: Sowohl ein Systemmodell als auch die ausführende Simulationsanwendung können zeitdiskret oder zeitkontinuierlich gestaltet sein [Dählmann, 2022]. Reale Systeme sind größtenteils zeitkontinuierlich, ihr Zustand ist also zu jedem beliebigen Zeitpunkt definiert und messbar. Die Zustände zeitdiskreter Systeme dagegen sind nur zu bestimmten, regelmäßig auftretenden Zeitpunkten definiert und feststellbar. [Bossel, 2004] Kontinuierliche Modelle basieren dabei in aller Regel auf stetigen mathematischen Funktionen und sind somit lückenlos definiert (vgl. Abbildung 28 rechts). Der Zustand zeitdiskreter Modelle hingegen verhält sich sozusagen sprunghaft und verändert sich lediglich zu den zuvor fest definierten Zeitpunkten (vgl. Abbildung 28 links). Computer arbeiten aufgrund ihrer Konstruktionsweise selbst immer zeitdiskret (d. h., es werden einzelne Rechenbefehle nacheinander ausgeführt, statt eine mathematische Funktion dauerhaft auszuwerten) [Tang et al., 2019], wodurch computersimulierte kontinuierliche Modelle streng genommen auch lediglich zeitdiskret dargestellt werden können. Ist diese notwendige Diskretisierung der Zeit aber in hinreichend kleine Schritte aufgeteilt, um das reale kontinuierliche Verhalten repräsentativ anzunähern, dann spricht man in der Regel trotzdem von kontinuierlichen Simulationen kontinuierlicher Systemmodelle. [Bossel, 2004]

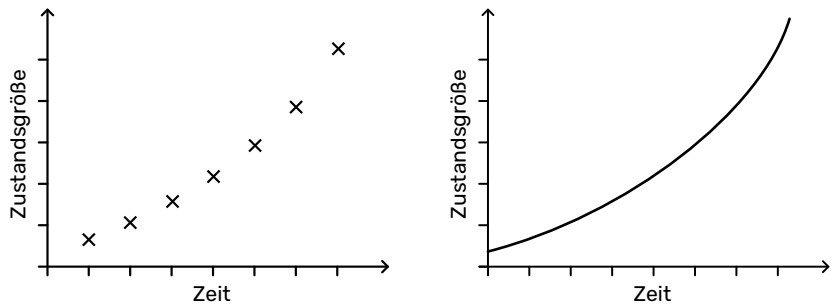


Abbildung 28: Ein mit der Zeit zunehmender Verlauf einer Zustandsgröße. Bei einer *zeitdiskreten Simulation* (links) (engl.: Discrete-Time Simulation, DTS) ist der Zustand nur zu diskreten Zeitpunkten definiert. Bei einem *zeitkontinuierlichen Modell* (engl.: Continous-Time Simulation, CTS) (rechts) ist der Zustand hingegen zu jedem beliebigen Zeitpunkt definiert, wodurch der Verlauf keine Lücken aufweist.

Der Vorteil zeitdiskreter Modelle ist vorrangig die mögliche Beschleunigung des Simulationslaufs, da Modellberechnungen lediglich an den zuvor definierten Zeitpunkten (z. B. einmal pro Sekunde) durchgeführt werden müssen und nicht für jeden beliebigen Punkt in der Zeit. Je größer die Intervalle zwischen den Zeitpunkten, die sogenannte Schrittweite (engl.: step size), sind, umso mehr Zeit ohne Berechnungen kann übersprungen und desto effizienter (schneller) kann der Simulationslauf durchgeführt werden. Dies geht allerdings einher mit einer abnehmenden Genauigkeit der Simulationsergebnisse, da eben jene Zeitspannen zwischen den Zeitpunkten übersprungen und infolgedessen mögliche Einflüsse in dieser Zeit ignoriert werden. Mit „Genauigkeit" ist in diesem Fall nicht die Genauigkeit einer einzelnen Berechnung gemeint, sondern die Realitätsnähe der Zustandstrajektorie als Ergebnis der Simulation. Die Bestimmung der Schrittweite einer zeitdiskreten Simulation bzw. eines zeitdiskreten Modells ist somit ebenfalls abhängig vom Zweck des Modells und somit vom konkreten Anwendungsfall und dem gewählten Abstraktionsgrad.

zustandsdiskret ⟷ zustandskontinuierlich: Es existieren Systeme und Prozesse, bei deren simulativer Betrachtung lediglich bestimmte Zeitpunkte, zu denen Ereignisse eintreten, von Interesse sind. Der Fertigungsprozess einer Montagelinie enthält bereits inhärent eine diskrete Ereignissteuerung: Ist eine Station der Montagelinie fertig mit der Bearbeitung des Werkstücks, so ist dieses für die nachfolgenden Stationen verfügbar und löst somit an diesen eine Zustandsveränderung aus. Die Zeitpunkte ohne Ereignisse dieser Art sind bei der Betrachtung logistischer Fragestellungen nicht relevant. Der Zustand von zustandsdiskreten Modellen ändert sich somit nur bei Eintreten von Ereignissen, weshalb eine solche Simulation auch *ereignisdiskrete Simulation* (engl.: Discrete-Event Simulation, DES) genannt wird. Die Ereignisse können zu beliebigen Zeitpunkten eintreten und eine sofortige Zustandsänderung auslösen. [Fujimoto, 2016] Zwischen dem Eintreten von zwei aufeinanderfolgenden Ereignissen wird der Zustand als konstant angenommen (vgl. Abbildung 29). Eine ereignisdiskrete Simulation hat den Vorteil, dass sie stark beschleunigt ausgeführt werden kann, da zwischen zwei Ereignissen keine Berechnungen durchgeführt werden [Lunze, 2017]. Ob ein zustandsdiskretes Modell sinnvoll einsetzbar ist, hängt wie auch die restlichen Modellierungsentscheidungen von dem Modellzweck und somit dem geplanten Anwendungsfall ab.

autonom ⟷ exogen getrieben: Systeme sind in der realen Welt immer in eine Umwelt eingebettet, die auf das System einwirkt (vgl. Abschnitt 6.3). Auf diese Einflüsse kann ein System reagieren und so exogen getrieben dynamisches Verhalten zeigen. Ein großer Teil des Systemverhaltens wird in der Regel allerdings systemintern durch gegenseitige Beeinflussung der Systemelemente und Zustandsgrößen und Rückkopplungsschleifen zwischen diesen erzeugt. Dieses intern entstandene Verhalten stellt die

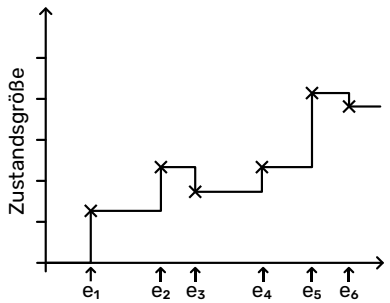


Abbildung 29: Beispielhafter Verlauf einer *zustandsdiskreten* Simulation (auch *ereignisdiskrete* Simulation genannt, engl.: Discrete-Event Simulation, DES). Die Markierungen an der x-Achse e_1 bis e_6 markieren den Zeitpunkt des Eintretens des jeweiligen Ereignisses.

systemcharakteristische Eigendynamik des Systems dar. Bei der Abbildung des Systems und seiner Umgebung durch ein oder mehrere Modelle kann die ein oder andere verhaltensbildende Seite stärker abstrahiert werden: Reagiert ein Modell ausschließlich auf externe Einflüsse, so ist es vollständig exogen getrieben. Erzeugt das Modell jedoch auch ohne ständige externe Einflüsse Veränderungen seines Zustands und somit dynamisches Verhalten, dann wird es in diesem Zusammenhang als *autonom* bezeichnet. [Bossel, 2004]

aggregiertes Verhalten ↔ individuelles Verhalten: Der Modellzweck bestimmt Art und Umfang der Modellbildung. Darunter kann auch fallen, dass für das Erreichen des Modellziels keine individuelle Betrachtung aller einzelner Akteure nötig ist. Dies ist besonders häufig der Fall bei der Simulation von Populationsentwicklungen, Verkehrsdichte, Verhalten von Menschenmengen, Entwicklung von Wäldern etc., also in Fällen, in denen das Verhalten der Masse als Ganzes statt das Verhalten des Individuums im Fokus steht. In solchen Fällen können die individuellen Zufälligkeiten der einzelnen Akteure durch statistische Mittelwerte angenähert und durch eine Gesamtdynamik beschrieben werden. In anderen Fällen, in denen das Verhalten eines einzelnen Akteurs Auswirkungen auf die Gesamtentwicklung der Menge haben kann, muss hingegen das Verhalten der Schlüsselakteure explizit modelliert und simuliert werden. [Bossel, 2004]

7.3.3 VERKEHRSSIMULATION

Verkehrssimulationen sind eine spezielle Form von Simulationsanwendungen, die Bewegungen und Interaktionen von Fahrzeugen und anderen Verkehrsteilnehmern nachbilden. Das häufigste Ziel ist dabei klassischerweise das Verständnis und die Vorhersage des Verkehrsaufkommens und -verhaltens, um Entscheidungsprozesse zu unterstützen. Derartige Modelle ermöglichen die Optimierung von Verkehrssystemen, die Bewertung der Auswirkungen neuer oder angepasster Infrastrukturen sowie die Verbesserung von Sicherheit und Effizienz. Die Anwendung von Simulationstechniken zur Untersuchung von Fragestellungen mit Bezug zu Verkehr und Transport begann bereits in den 1950er Jahren [Wigan, 1970]. Vor allem durch die exponentiell gesteigerte Leistung von Computer-Hardware hat der Einsatz von Simulationsanwendungen seitdem weiter an Relevanz gewonnen und infolgedessen eine Fülle verschiedener Ausprägungen und Einsatzzwecke hervorgebracht. In jüngerer Zeit haben Verkehrssimulationen, aus oftmals rein wissenschaftlicher Anwendung heraus, auch Einzug in die Praxis gefunden [Barceló, 2010a]. Spätestens seit dem Aufkommen von computergenerierten dreidimensionalen Bewegtbildern (Computer Generated Image Visual System, CGIVS) zur Darstellung virtueller Außenansichten für Flugsimulatoren und dem umfassenden Einsatz solcher Simulatoren in der Ausbildung von angehenden Piloten ist der Begriff *Simulator* in der Gesellschaft angekommen.

Außerhalb von Ausbildungen zukünftiger Fahrzeugführer wird wohl am häufigsten Straßenverkehr simuliert: Zur Planung von Anpassungen oder Erweiterungen des Verkehrssystems ist die simulative Überprüfung der möglichen Maßnahmen ein zentrales Werkzeug geworden. Durch die Beantwortung von Fragen wie „Wirkt sich ein zusätzlicher Kreisverkehr besser auf den Verkehrsfluss aus oder ist eine angepasste Ampelschaltung ausreichend?“, „Kann eine Erweiterung der örtlichen Hauptverkehrsstraße um eine Spur pro Fahrtrichtung dem täglichen Stau zur Feierabendzeit entgegenwirken?“ und „Wie wirkt sich autonomes Fahren auf das örtliche Verkehrssystem aus?“ können potenziell kostspielige Maßnahmen im Voraus auf Effizienz und Effektivität überprüft werden. Auch die in dieser Arbeit im Fokus stehenden Simulationen maritimen Verkehrs sind schon lange Untersuchungsgegenstand wis-

senschaftlicher Arbeiten [Davis et al., 1980]. Da Simulationsanwendungen und -modelle von Straßenverkehr deutlich präsenter und vielfältiger sind, wird im Nachfolgenden trotzdem mehrheitlich auf Veröffentlichungen und Beispiele aus dieser Domäne Bezug genommen.

7.3.3.1 BETRACHTUNGSEBENE

Bei klassischen Fragestellungen liegt der Fokus häufig auf aggregierten Werten, wie der Anzahl an Fahrzeugen pro Zeiteinheit, der durchschnittlichen Geschwindigkeit oder der durchschnittlichen Wartezeit. In solchen Fällen ist es für entsprechende simulative Untersuchungen ausreichend, die Gesamtheit der Verkehrsteilnehmer als Ganzes zu betrachten. Ein entsprechendes, stark abstrahiertes Modell ist in der Lage, die Komplexität des Gesamtsystems *Verkehr* auf ein Minimum zu reduzieren und die von einem Simulationslauf benötigte Zeit stark zu verkürzen [Ghadai et al., 2016]. Ein erstes solches Verkehrsfluss-Modell fand im Jahr 1955 breite Beachtung: Lighthill & Witham untersuchten mit Hilfe einer Methode zur Beschreibung kinematischer Wellen die Beziehung zwischen Durchfluss, Konzentration und Ausbreitung von Verkehr auf langen, stark befahrenen Hauptverkehrsstraßen [Lighthill & Witham, 1955]. Mit der bereits erwähnten Zunahme der Rechenleistung verschiebt sich in jüngerer Zeit der Interessenschwerpunkt aber zunehmend hin zu kleinteiligeren Verkehrssimulationen, welche jeden Verkehrsteilnehmer oder sogar Fahrzeugkomponenten individuell abbilden [Barceló, 2010a].

Um den unterschiedlichen Anforderungen möglicher Anwendungsfälle gerecht zu werden, hat sich über die Jahre eine Einteilung der Modelle in drei Granularitätsebenen etabliert: makro-, mikro- und submikroskopische Verkehrsmodelle (vgl. Abbildung 30). Die Reinformen dieser werden nachfolgend genauer betrachtet. Mehrere Ebenen können aber auch in einem Simulationsmodell miteinander kombiniert werden. Ein Beispiel einer solchen Mehrebenen-Verkehrssimulation zur Untersuchung von städtischem Güterverkehr unter Betrachtung lang-, mittel- und kurzfristiger Einflüsse ist in [Alho et al., 2017; Adnan et al., 2016] zu finden.

Makroskopische Verkehrsmodelle & -simulationen

Makroskopische Verkehrsmodelle sind die am stärksten abstrahierenden der drei genannten. Sie aggregieren das individuelle Verhalten und die individuelle Dynamik eines Verkehrsteilnehmers zu einer Einheit [Mohan & Ramadurai, 2013]. Der Zustand des Gesamtsystems Verkehr kann durch nur wenige Variablen wie Verkehrsfluss, Verkehrsdichte und mittlere Geschwindigkeit beschrieben werden [Mohan & Ramadurai, 2013; Treiber & Kesting, 2010]. Zur Beschreibung des aggregierten dynamischen Verhaltens sind zeitkontinuierliche, gleichungsbasierte Ansätze dabei am gebräuchlichsten [Gütlein,

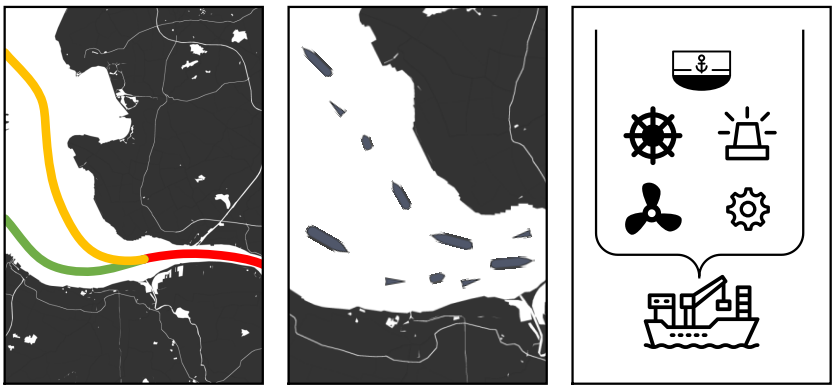


Abbildung 30: Visualisierung der drei etablierten Granularitäten von Verkehrssimulationen: makroskopisch, mikroskopisch, submikroskopisch (v. l. n. r.)

2023], welche zumeist von Beschreibungs- und Modellierungsansätzen aus dem Gebiet der Hydrodynamik inspiriert sind (vgl. u. a. [Lighthill & Witham, 1955]), weswegen sie auch gelegentlich als hydrodynamische Modelle bezeichnet werden [Treiber & Kesting, 2010].

Makroskopische Modelle eignen sich am besten für kurzfristige Prädiktionen, die Untersuchung aggregierter Werte sowie als Grundlage für Verkehrsmanagementsysteme. Die Detailarmut bietet zudem Vorteile in Bezug auf Simulationen: Für die Durchführung eines Simulationslaufs werden nur wenige Eingabedaten benötigt und der Rechenaufwand ist gering, da lediglich eine einzelne aggregierte Entität am Simulationslauf teilnimmt und das Verhalten größtenteils durch stochastische und statistische Prozesse gesteuert wird. [Gütlein, 2023]

Beispiele für makroskopische Simulationen sind u. a. die kommerzielle Verkehrsplanungssoftware *PTV Visum*[37] sowie die in Python geschriebene Open-Source Verkehrsnetzsimulation *UXsim*[38].

Mikroskopische Verkehrsmodelle & -simulationen

Den makroskopischen Modellen stehen direkt die detaillierteren mikroskopischen Modelle gegenüber. Diese haben als primären Betrachtungs- und Modellierungsgegenstand die individuellen Verkehrsteilnehmer und ihr jeweiliges individuelles Verhalten in Abhängigkeit von ihrer jeweiligen Umgebung [Gütlein, 2023; Mohan & Ramadurai, 2013; Treiber & Kesting, 2010]. Es werden also die Absichten, die Entscheidungsfindung und die anschließenden Handlungen individuell pro Verkehrsteilnehmer betrachtet und modelliert [Sykes, 2010]. Aktionen wie das Beschleunigen oder Verlangsamen werden dabei innerhalb einer Simulation in der Regel von jedem Fahrzeug individuell über seine Zustandsgrößen selbst durchgeführt [Gütlein, 2023]. Eine Abbildung physikalischer Prozesse oder des Zusammenspiels einzelner Komponenten innerhalb eines Fahrzeugs wird hier nicht vorgenommen. Obwohl der Haupteinsatzzweck mikroskopischer Modelle die detailliertere Untersuchung von individuellem Verhalten und emergenten Eigenschaften ist, können auch makroskopische Untersuchungen durchgeführt werden: Aggregierte Größen, die den Gesamtzustand des Verkehrs beschreiben, können *bottom-up* inferiert werden [Gütlein, 2023].

Mit dem höheren Detailgrad geht eine höhere Komplexität einher und die Erstellung mikroskopischer Modelle gestaltet sich somit aufwendiger als im makroskopischen Bereich. Zusätzlich führt eine gesteigerte Komplexität immer zu einer höheren Anfälligkeit für Modellierungsfehler [Gütlein, 2023]. Da das Verhalten einer Vielzahl von Entitäten simuliert werden muss, ist zudem der Rechenaufwand höher und der Umfang somit stärker begrenzt, als es bei Simulationen makroskopischer Modelle der Fall ist [Mohan & Ramadurai, 2013]. Aus dem gleichen Grund sind zur Durchführung eines mikroskopischen Simulationslaufs deutlich mehr Eingabedaten notwendig: Jedes Fahrzeug muss definiert und parametriert sein und über eine Route oder ein Ziel verfügen.

Ein Beispiel für eine mikroskopische Simulationsanwendung ist das Open-Source Software-Paket *Simulation of Urban Mobility (SUMO)*[39]. [Krajzewicz, 2010]

Submikroskopische Verkehrsmodelle & -simulationen

Die Kategorie der detailliertesten Modelle ist gleichzeitig auch die jüngste und entsprechend begrifflich noch nicht vollständig einheitlich definiert. So findet man in aktueller wissenschaftlicher Literatur

[37] `https://www.ptvgroup.com/de/produkte/ptv-visum` [letzter Zugriff: 17.02.26]
[38] `https://github.com/toruseo/UXsim` [letzter Zugriff: 17.02.26]
[39] `https://eclipse.dev/sumo/` [letzter Zugriff: 17.02.26]

ebenfalls die Bezeichnungen *nanoskopisch* und *komponentenbasiert*. Auch inhaltlich existieren verschiedene Interpretationen dieser Kategorie: Einige Autoren definieren die Zugehörigkeit über einen gesteigerten Realismusgrad, der z. B. mit der Integration dynamischer Physiksimulation erreicht werden kann, oder über die Integration einer detaillierten dreidimensionalen Umgebung, um u. a. die Modellierung optischer Sensoren zu ermöglichen [Gütlein, 2023]. Andere sehen mehr eine weitere Unterteilung des Betrachtungsgegenstandes, wie es auch beim Schritt von makro- zu mikroskopisch der Fall ist. Individuell modelliert würden hier also einzelne Komponenten eines Fahrzeugs, die miteinander agieren und so das Verhalten des Fahrzeugs bestimmen. Dabei ist der Detailgrad individuell auf den Anwendungsfall abzustimmen (vgl. Abschnitt 7.3.1) und kann somit von der Modellierung eines Fahrers, der das Fahrzeug bedient [Ni, 2003], bis hin zur Modellierung einzelner technischer Komponenten des Fahrzeugs gehen. Eine komponentenbasierte Modellierung ermöglicht die Simulation und Untersuchung einzelner Komponenten als Teil eines Systemverbunds sowie auch die Untersuchung emergenten Verhaltens, was die Art der Modellierung für die Entwicklung automatisierter Fahrsysteme interessant macht.

Mit dem abermals gesteigerten Detailgrad gehen die Nachteile von mikroskopischen Modellen in gesteigerter Form einher. Vor allem die dynamische Simulation von physikalischen Abläufen und die Berechnung und Darstellung einer dreidimensionalen Umgebung benötigen viel Rechenleistung. Häufig werden derartige Modelle und Simulationen eingesetzt, um Vorgänge innerhalb eines Fahrzeugs zu untersuchen, während es sich im Straßenverkehr bewegt, weshalb zur Reduzierung des Aufwands und der Rechenlast alle Fahrzeuge, die den Umgebungsverkehr für das zu untersuchende Fahrzeug bilden, weniger detailliert oder sogar mikroskopisch modelliert und simuliert werden können.

7.3.3.2 AGENTENBASIERTE SIMULATIONEN

Eine spezielle Form der Modellierung und Simulation, die nicht explizit in Abschnitt 7.3.2 betrachtet wurde, ist die agentenbasierte Modellierung (engl.: Agent Based Modelling, ABM). Dieser Ansatz ist zu Teilen als übergreifend zu den genannten Kategorien zu verstehen, da agentenbasierte Modelle wiederum unterschiedlicher Ausprägung sein können. ABM stellt den Ansatz dar, dynamisches Verhalten komplexer Systeme dadurch abzubilden, dass die einzelnen Einheiten[40] dieses Systems auf Basis einfacher Interaktionsregeln mit ihrer Umgebung und untereinander interagieren [Gütlein, 2023; Ghadai et al., 2016].

Russell und Norvig beschreiben einen Agenten als „[…] alles, was seine Umgebung mit Hilfe von Sensoren wahrnehmend und auf diese Umgebung mit Hilfe von Aktoren einwirkend betrachtet werden kann“[41] [Russell & Norvig, 2016]. Anhand dieser Definition ist zu erkennen, dass Fahrzeuge die Kriterien von Agenten erfüllen. Sensoren, wie Light Detection and Ranging (LiDAR) oder im übertragenen Sinne die Augen des Fahrzeugführers, nehmen die Umgebung des Fahrzeugs wahr, elektronische Systeme oder der Mensch selbst verarbeiten diese Daten, leiten daraus Entscheidungen ab, das Fahrzeugsystem verändert seinen internen Zustand und beeinflusst seine Umgebung durch Aktuatoren, indem z. B. eine neue Zielgeschwindigkeit definiert und die Geschwindigkeit entsprechend zur Erreichung dieser durch Beschleunigung oder Verzögerung verändert wird. Im Kontext der virtuellen Umgebung eines Simulationslaufs werden die Daten nicht von realen Sensoren erzeugt, sondern aus der gemein-

[40] Eine Übersicht darüber wann eine Einheit ein Agent ist, was einen Agenten von einem Akteur unterscheidet und eine historische Einordnung der Entstehung und Nutzung beider Begriffe findet sich in [Schulz-Schaeffer, 1998].

[41] Übersetzt aus dem Englischen; Auslassung;

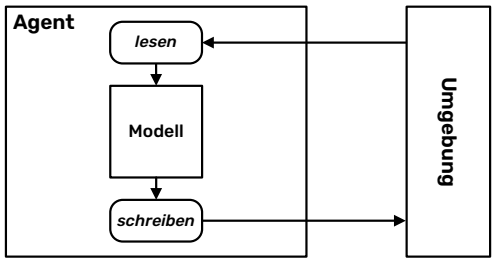


Abbildung 31: Abstrakte Darstellung eines einfachen nicht näher konkretisierten Agenten als Teil eines Simulationslaufs (angelehnt an [Russell und Norvig, 2016]). Das dynamische Verhalten des Agenten wird durch das interne Modell bestimmt, welches die Rolle der von Russell und Norvig als *Agent Program* bezeichneten Implementierung einnimmt.

samen Umgebung der Agenten ermittelt. Daher können die Begriffe *Sensor* und *wahrnehmen* sowie *Aktor* und *agieren* im Kontext von Simulationsanwendungen durch die informationstechnischen Begriffe *Lesen* und *Schreiben* ersetzt werden: Ein Agent liest die Werte aus dem Modell der gemeinsamen Umgebung (Eingangsgrößen) und schreibt Veränderungen an der Umgebung durch das eigene Handeln (Ausgangsgrößen) in diese zurück (vgl. Abbildung 31).

Sind mehrere interagierende Agenten Teil eines Simulationslaufs, wird auch von Multi-Agenten-Simulationen (MAS) gesprochen. MAS haben ihren Weg innerhalb der letzten Jahrzehnte in unterschiedlichste wissenschaftliche Domänen gefunden [Drogoul et al., 2003], was hauptsächlich auf drei Eigenschaften zurückzuführen ist: (1) Die Flexibilität hinsichtlich der Anwendbarkeit des Ansatzes zur Modellierung von Individuen [Drogoul et al., 2003]; (2) Die Möglichkeit unterschiedliche Repräsentationsebenen innerhalb eines einheitlichen konzeptionellen Rahmens zu handhaben [Drogoul et al., 2003]; (3) Die Modellierung komplexer Systeme (z. B. eines Verkehrssystems) gestaltet sich sehr natürlich, da die Komponenten des Systems (z. B. die einzelnen Verkehrsteilnehmer) in der Realität ebenfalls voneinander getrennt sind und nur lose gekoppelt über Wahrnehmung, Kommunikation und Aktionen miteinander agieren. Agentenbasierte Modellierung ermöglicht es also, die natürlichen Strukturen direkt abzubilden [Kasereka et al., 2023] und Komplexität bereits auf konzeptioneller Ebene handhabbar zu machen [Ghadai et al., 2016].

Nach Hanappi [2017] muss eine agentenbasierte Simulation folgende drei Eigenschaften besitzen:

Agenten Struktur: Die innere Struktur eines Agenten muss programmatisch so beschrieben werden, dass die folgende Sequenz abgearbeitet wird: (1) Lesen von Werten aus der Umgebung, (2) Überführung der Eingangsgrößen in eine interne Zustandshaltung, (3) Wahl einer Aktion basierend auf dem aktuellen internen Zustand.

Heterogenität: Im Rahmen von Multi-Agenten-Simulationen ist zu gewährleisten, dass die internen Modelle der Agenten heterogene Ausprägungen aufweisen können.

Umgebung: Es muss ein zentrales programmatisches Modell geben, das selbst keinen Agenten darstellt. Dieses Modell stellt den Zustand der Umwelt zur Verfügung, innerhalb derer die Agenten ihre Umweltdynamik wahrnehmen und mittels der von ihnen gewählten und ausgeführten Aktionen auf diese Einfluss nehmen können.

Die konkrete Ausgestaltung des internen Modells eines Agenten ist dabei bewusst nicht spezifischer beschrieben. So ist es u. a. möglich, komplexere Systeme inklusive ihrer Komponenten im Sinne eines submikroskopischen Modells durch einen Agenten abzubilden (vgl. Abbildung 32). Die Vorteile eines solchen Modells, Komplexität handhabbarer zu machen und die Möglichkeit zur Untersuchung

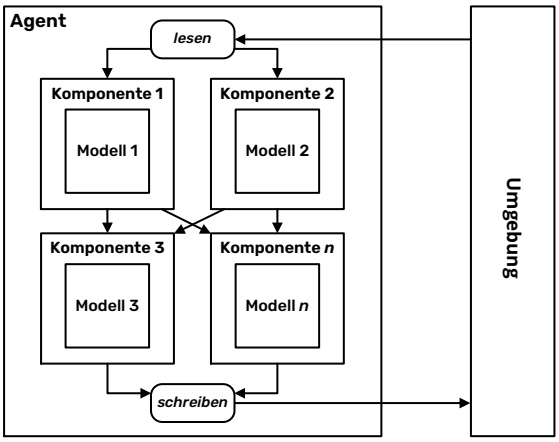


Abbildung 32: Abstrakte Darstellung eines Agenten als Repräsentation eines System-of-Systems (vgl. Abbildung 31)

emergenter Eigenschaften durch explizite Modellierung systeminterner Interaktionen, lassen sich somit auch auf agentenbasierte Modelle übertragen. Moderne Schiffe als *System-of-Systems* (SoS) (vgl. Abschnitt 6.3) könnten daher ebenfalls bei gleichzeitiger individueller Modellierung der Komponenten durch einen einzelnen Agenten repräsentiert werden.

Der Nachteil der höheren Gesamtkomplexität der Modelle und der damit einhergehende Anstieg des Rechenbedarfs, welcher im Abschnitt 7.3.2 bereits für mikroskopische und submikroskopische Simulationsmodelle festgestellt wurde, gilt analog auch für agentenbasierte Modelle und Simulationen [Ghadai et al., 2016]. Ein Beispiel für eine agentenbasierte mikroskopische Simulationsanwendung zur Untersuchung von Straßenverkehrsflüssen in geografisch großangelegten Gebieten mit einer großen Anzahl teilnehmender Agenten ist das *Open-Source-Framework Multi-Agent Transport Simulation (MATSim)*[42]. Aufgrund des Ziels, mit dem MATSim entwickelt wurde, Agentenmengen von mehreren tausend bis zu mehreren Millionen innerhalb eines Simulationslaufs zu unterstützen, wurde von den Entwicklern auf ein warteschlangenbasiertes Modell gesetzt statt auf eine vollumfängliche physikalische Simulation oder ein komponentenbasiertes Modell. Agentenbasierte-Simulationen bieten durch ihre starke Trennung der Teilmodelle in Form der Agenten aber über monolithische mikroskopische Simulationen die Möglichkeit einer Aufteilung der Berechnungen auf physikalisch voneinander unabhängige Computersysteme und somit eine Verteilung der Rechenlast.

Agentenbasierte Modelle sind zumeist programmatische Modelle und entsprechend zeitdiskret (vgl. Abschnitt 7.3.2). Das Ergebnis der Simulation eines agentenbasierten Verkehrsmodells ist in diesen Fällen ein Tupel von Mengen der Systemzustände aller Agenten zum diskreten Zeitpunkt $t \in \{t_1, \ldots, t_{end}\}$. Dieses Tupel entspricht der Trajektorie des übergreifenden Verkehrssystems und wird hier als *Verkehrstrajektorie* bezeichnet. Eine einzelne Menge aus diesem Tupel beschreibt den Zustand des Verkehrs zu einem bestimmten Zeitpunkt und wird hier als *Verkehrslage* bezeichnet. Sowohl Verkehrstrajektorien als auch einzelne Verkehrslagen können von großer Wichtigkeit für den Einsatz von Verkehrssimulationen als Testwerkzeug sein: Durch ihre Analyse kann, nachdem der jeweilige Simulationslauf bereits durchgeführt wurde, über Erfolg oder Misserfolg des jeweiligen Tests entschieden werden. Auch über den Einsatz als Testwerkzeug hinaus kann eine solche simulierte Verkehrsentwicklung von Nutzen sein: u. a. als alternativer Eingabewert von Verkehrsmanagementsystemen im Rahmen von Ausbildungen zukünftigen Personals statt historischer, durch Aufzeichnung gewonnener

[42] `https://www.matsim.org/` [letzter Zugriff: 17.02.26]

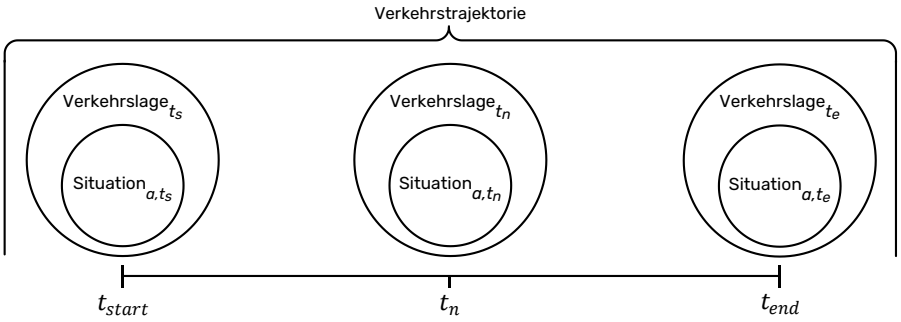


Abbildung 33: Ein Tupel von *Verkehrslagen* als Ergebnis eines zeitdiskreten (sub-)mikroskopischen Simulationslaufs

Daten realen Verkehrs. Zusätzlich können Verkehrslagen als Startszenen von Szenarien als Input zukünftiger Simulationsläufe dienen. Der Zusammenhang der Simulationszeit, Verkehrslagen und der Verkehrstrajektorie als Ergebnis eines Simulationslaufs ist in Abbildung 33 dargestellt. Um den inhaltlichen Zusammenhang mit den Konzepten der Szene und der Situation zu verdeutlichen, ist zusätzlich eine Situation, wie sie in Abschnitt 7.2.4 definiert wurde, als Teilmenge einer Verkehrslage dargestellt.

7.3.4 X-IN-THE-LOOP TESTVERFAHREN

Damit eine Verkehrssimulationsanwendung überhaupt für Testläufe von Modell-, Software- und Hardwarekomponenten eingesetzt werden kann, ist es nötig, diese an dem Simulationslauf teilnehmen zu lassen. Sie müssen in das Simulationsmodell integriert, genauer gesagt adäquat an dieses angebunden werden, um Informationen aus dem aktuellen Zustand des Simulationsmodells als Eingangsgrößen zu verarbeiten und Ausgangsgrößen zurück in das Simulationsmodell zu geben. Beide Datenflussrichtungen sind dabei essenziell für eine interaktive Teilnahme am Simulationslauf: (1) Der Zugriff auf Zustandsdaten des Simulationsmodells ermöglicht es der zu testenden Komponente, auf das Verhalten simulierter Entitäten zu reagieren, (2) die Beeinflussung des Simulationsmodells ist nötig, damit das Verhalten der Komponente im Simulationsmodell wiedergespielt wird und es den simulierten Entitäten ermöglicht wird, auf das Verhalten der Komponente zu reagieren (vgl. Abbildung 34).

Die gewählte Anordnung der Elemente dieser bidirektionalen Beziehung lässt eine Feedbackschleife (engl.: feedback loop) erkennen, woraus sich der geläufige Name dieser Testmethode ergibt: *X-in-the-Loop (XiL)* (vgl. u. a. [Karthaus et al., 2021; Stefan Riedmaier et al., 2018; Tibba et al., 2016; Düser, 2010]). Das X steht dabei stellvertretend für die Abstraktionsebene der Einheit, deren Funktionalität überprüft werden soll. Dabei haben sich vier konkrete Ansätze als die üblichen hervorgetan: *Model-in-the-Loop (MiL)*, *Software-in-the-Loop (SiL)*, *Hardware-in-the-Loop (HiL)* und *Vehicle-in-the-Loop (ViL)* [Jesenski et al., 2019; Brinkmann & Hahn, 2017; Pfeffer & Leichsenring, 2016]. Ferner werden immer mal wieder weitere Ansätze entwickelt, die sich zwischen den vier etablierten Ebenen einordnen, über diese hinausgehen oder mehrere der klassischen Ebenen umfassen, aber noch keine weite Verbreitung gefunden haben, wie *Dynamic Vehicle-in-the-Loop (DynViL)* [Drechsler et al., 2022] und *Scenario in the Loop (SCiL)* [Varga et al., 2021]. Im Kontext automatisierter Fahrfunktionen hat sich allen anderen Ansätzen vorgelagert zusätzlich die Idee des *Concept-in-the-Loop (CiL)* herausgebildet.

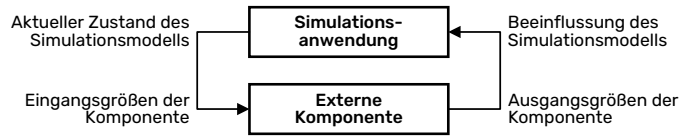


Abbildung 34: Bidirektionale Einbindung einer zu testenden Komponente in einen Simulationslauf

Tabelle 5: Betrachtungsebene des Simulationsmodells (Zeilen), die Art der möglichen Integration des jeweiligen Testobjekts in einen Simulationslauf (Spalten) und die Art von geeigneten Testobjekten (Zellen).

	MiL	**SiL**	**Hil**	**ViL**
Mikroskopisch	System	-	-	System
Submikroskopisch	*System* Teilsystem	-	Teilsystem	*System*
Komponentenbasiert	*System* *Teilsystem* Komponente	-	*Teilsystem* Komponente	*System*
Subkomponentenbasiert	*System* *Teilsystem* *Komponente* Software-Einheit	Software-Einheit	*Teilsystem* *Komponente*	*System*

Um den Entwicklungsprozess von Systemen und Systemkomponenten für automatisierte Fahrzeuge durchgehend unterstützen zu können, ist die Möglichkeit, die (Zwischen-)Ergebnisse aller Stufen des Entwicklungsprozesses (vgl. Abbildung 19) an Simulationsläufen teilnehmen zu lassen, obligatorisch. Gleichzeitig wurde in Abschnitt 7.3.1 die Relevanz der präzisen Bestimmung des Modellzwecks begründend hervorgehoben. Die sich dadurch ergebene starke Varianz des Detailgrads von Simulationsmodellen wurde unter anderem in Abschnitt 7.3.3 in Bezug auf Verkehrssimulationen erneut aufgegriffen. Die somit sowohl auf der Seite der zu testenden Komponente als auch auf der Seite der Simulationsanwendung bestehende Möglichkeit, einem von mehreren Abstraktionsniveaus zu entsprechen, führt zu einer Vielzahl unterschiedlicher Kombinationen, von denen nicht alle für aussagekräftige Testläufe geeignet sind.

Während ein stark abstraktes Modell ebenfalls nur eine stark abstrakte Darstellung der Umgebung und des Verhaltens der dynamischen Akteure während des Simulationslaufs erzeugt, kann ein detaillierteres Simulationsmodell eine detailliertere Ausgabe erzielen [Hanke, 2020]. Da die zu testende Komponente ihre Eingangsgrößen während des Simulationslaufs aus dem aktuellen Zustand des Simulationsmodells bezieht, muss das Modell den Erwartungen der zu testenden Komponenten entsprechend granulare Daten liefern. Hanke hat in diesem Zusammenhang für die Automobil-Domäne drei Granularitätsebenen identifiziert, die dem traditionellen V-Modell der Softwareentwicklung entliehen wurden: System, Teilsystem und Komponente. Im ersten Fall wird das gesamte integrierte Fahrzeug untersucht, sodass eine einzige Verhaltensfunktion als Modellabstraktion für jeden simulierten Verkehrsteilnehmer ausreicht. Teilsysteme als Ergebnis eines Zwischenschritts der Entwicklung müssen in einen simulierten Verkehrsteilnehmer eingebettet werden, wodurch die Teilsysteme, die auf das Verhalten des zu Testenden einwirken könnten, explizit modelliert werden müssen. Auf der kleinteiligsten von Hanke genannten Ebene wird jede einzelne Komponente eines Systems betrachtet und entsprechend muss das Simulationsmodell auch Systemkomponenten und Interaktionen zwischen diesen vorsehen. Um auch die kleinsten Teile eines gängigen Softwareentwicklungsprozesses abdecken zu können, wird hier zusätzlich eine vierte Ebene eingeführt, die einzelne Software-Funktionseinheiten innerhalb einer Komponente betrachtet (s. dazu Abbildung 14 in Abschnitt 6.3 und [Reiher & Hahn, 2021b]).

Welche Granularität ein Simulationsmodell aufweisen muss, damit Testobjekte der genannten Betrachtungsebenen in den unterschiedlichen Phasen des Entwicklungsprozesses sinnvoll in einen Simulationslauf eingebunden werden können, ist in Tabelle 5 dargestellt. Einige Testobjektausprägungen lassen

eine Anbindung an Simulationsmodelle mehrerer Granularitäten zu. In diesem Fall ist das am wenigsten komplexe Simulationsmodell zu bevorzugen: Die arbeitsintensiveren Alternativen sind kursiv gedruckt. Zusätzlich sei an dieser Stelle darauf hingewiesen, dass die Wahl einer der kursiv gedruckten Alternativen keinesfalls falsch ist, sondern auch von den genauen Anforderungen des konkreten Testobjekts und des konkreten Testfalls abhängt. Beide sind daher relevant für die Definition des Modellzwecks (vgl. Definition von *Modell* in Abschnitt 7.2.4).

7.4 MODELLIERUNG VON SZENARIEN

Um Verkehrssimulationen zur Durchführung szenariobasierter Testläufe einsetzen zu können, ist eine persistente und eindeutige Repräsentation des Szenarios in einer Art und Weise nötig, die von der Simulationsanwendung gelesen und verstanden werden kann (vgl. [Ma et al., 2021; Reiher & Hahn, 2021b]). Für das Artefakt, welches als Eingangsgröße einer Simulationsanwendung dienen soll, wird in dieser Arbeit fortan der Begriff *Szenarioinstanz* verwendet, um eine klare Trennung von *Szenariobeschreibungen* zu schaffen, die eher als Hilfsmittel zur Kommunikation dienen, wie linguistische Formulierungen. Eine solche Szenarioinstanz ist zudem klar getrennt zu betrachten von dem Simulationsmodell (vgl. Abschnitt 7.3.1) und der konkreten Instanziierung dessen im Rahmen eines Simulationslaufs, welche u. a. zusätzlich vergangene Zustände in einer Historie halten (vgl. [Altiok & Melamed, 2007]), globale Funktionen wie solche zur Berechnung grundlegender physikalischer Eigenschaften und Gesetze bereitstellen sowie die Basis für die grundlegenden Funktionen eines Simulationslaufs bilden kann. Die Szenarioinstanz hingegen versteht sich in diesem Zusammenhang lediglich als eine Kombination aus parametrierten Ausgangszuständen (vgl. Definition einer *Szene* in Abschnitt 7.2.4) sowie Parametern und Systemanregungen (vgl. Definition eines *Szenarios* in Abschnitt 7.2.4), die als Eingabewerte für die Durchführung einer Simulation dienen. Wie diese Artefakte des szenariobasierten Simulationsprozesses miteinander in Beziehung stehen, ist in Abbildung 35 dargestellt. Die Eigenschaften dieser und deren Beziehungen zueinander werden aufgrund ihrer zentralen Rolle für den effektiven szenariobasierten Einsatz von Simulationen nachfolgend genauer betrachtet.

7.4.1 SIMULATIONSMODELLE

Jede Simulation dynamischen Verhaltens setzt zunächst die Erstellung eines Modells voraus, das eben jenes Verhalten beschreibt [Mattern, 1996] (vgl. Abschnitt 7.2.4). Da in den wenigsten Fällen lediglich das dynamische Verhalten eines isolierten Systems oder einer isolierten Systemkomponente untersucht werden soll (vgl. Abschnitt 7.2, Abschnitt 7.3.3), ist zusätzlich die Modellierung der Umgebung

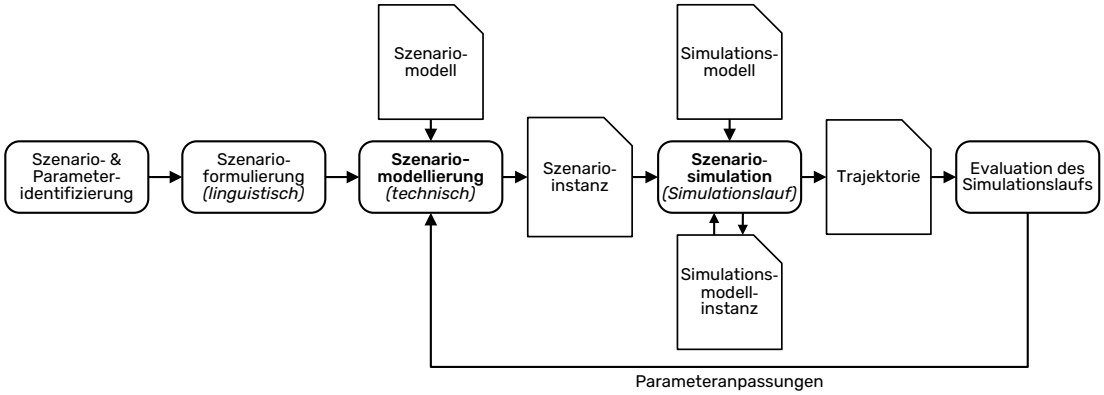


Abbildung 35: Die Modelle und Artefakte eines szenariobasierten Simulationslaufs (vgl. Abbildung 2)

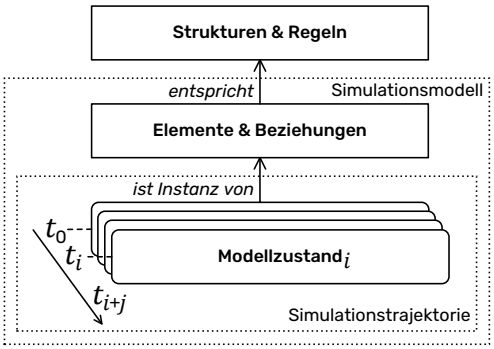


Abbildung 36: Beziehungen des Simulationsmodells einer monolithischen Simulationsanwendungen

notwendig. Die Umgebung umfasst dabei in der Regel sowohl stationäre Elemente wie Küstenlinien oder Verkehrszeichen und dynamische Elemente wie Wetterphänomene oder Bojen als auch weitere Akteure in Form von weiteren Verkehrsteilnehmern (vgl. Abschnitt 7.2.4).

Der klassische Ansatz zur Initiierung von Simulationsläufen besteht darin, in einer Simulationsanwendung ein abstraktes Simulationsmodell zu integrieren (Implementierungen der Entitäten und ihrer Beziehungen in Form von ausführbarem Programmcode), welches die möglichen statischen und dynamischen Inhalte des Simulationslaufs repräsentiert. Die für den konkreten Simulationslauf notwendigen Bestandteile dieses abstrakten Modells werden dann von der Simulationsanwendung durch Instanziierung sowie Parametrierung in eine konkrete Variante des Modells überführt. Diese instanziierte Teilmenge des Modells stellt den Simulationszustand zum Simulationszeitpunkt t_0 (Ausgangszustand) dar. Mit diesem als Basis wird der Simulationslauf gestartet. Während des Simulationslaufs verändert sich der Zustand der Modellinstanz und eine Simulationstrajektorie entsteht.

Zur Implementierung einer Simulationsanwendung und eines integrierten Simulationsmodells wird in der Regel eine höhere prozedurale oder objektorientierte Programmiersprache wie C, C++, C# oder Java eingesetzt. Die Syntax einer solchen Programmiersprache ist formalisiert und durch eine Menge von Regeln festgelegt [Ernst et al., 2016]. Eine Programmiersprache gibt somit unter anderem vor, wie der Programmcode strukturiert sein muss, was für Operationen möglich sind und welche Datenstrukturen verwendet werden können. Eine Programmiersprache gibt so einen fest definierten Rahmen vor, innerhalb dessen sich der Programmierer bewegen muss. Wird bei der programmatischen Umsetzung eines Simulationsmodells zusätzlich gewissenhaft objektorientiert gearbeitet, so stellt der Programmierer sehr wahrscheinlich zusätzlich Regeln auf, an die er sich bei der Umsetzung des eigentlichen Simulationsmodells halten muss, indem er zunächst einheitliche Strukturen durch die Nutzung von Konstrukten wie Schnittstellen und abstrakten Klassen definiert. Diese strukturellen Einschränkungen und Regeln haben einen maßgeblichen Einfluss auf den Aufbau eines Simulationsmodells, wodurch sich eine enge Verknüpfung ergibt (vgl. Abbildung 36).

7.4.2 SZENARIOSPEZIFIKATIONSSPRACHEN & -MODELLE

Um dem Anspruch der Effizienz an den szenariobasierten Testprozess gerecht zu werden, müssen die zu testenden Szenarien eindeutig interpretierbar und wiederverwendbar sein. Daraus ergibt sich die Notwendigkeit, die zu testenden Szenarien strukturiert und persistent definieren zu können. Insbesondere im Hinblick auf die zentrale Idee der szenariobasierten V&V, identifizierte Szenarien zu katalogisieren und zu späteren Zeitpunkten wiederholt wiederverwenden zu können. Auch für die hier gewünschte simulative Ausführung der Testszenarien ist dieser Punkt von hoher Bedeutung.

Ein Fahrszenario besteht in der Regel aus verschiedenen statischen und dynamischen Elementen (vgl. Abschnitt 7.2.4). In der Domäne des Straßenverkehrs gehören dazu u. a. die Umgebung wie Straßenbreiten, Anzahl der Fahrspuren, Straßenkrümmung, Verkehrszeichen, Fußgängerwege, die aktiven Elemente wie eine Reihe von Fahrzeugen, Fußgänger, Ampeln inkl. Ausgangszuständen und Verhaltensbeschreibungen sowie die in dieser Umgebung zu testende Funktionseinheit [Ma et al., 2021]. Je nachdem, für welchen Zweck und durch wen Szenarien definiert werden, können sie unterschiedlich detailliert ausfallen. Menzel et al. definieren drei Abstraktionsniveaus für Szenariospezifikationen: funktionale, logische und konkrete Szenarien [Menzel et al., 2018].

Funktionale Szenarien stellen die abstrakteste Form dar und werden in der Regel linguistisch beschrieben, damit menschliche Domänen-Experten diese einfach verstehen, mithilfe von ihnen kommunizieren, über sie diskutieren und neue erstellen können [Menzel et al., 2018]. Sie werden häufig in sehr frühen Phasen der Systementwicklung oder zu Beginn des Entwurfs des Testprozesses genutzt und bilden die Grundlage für eine größere Anzahl weniger abstrakter Szenarien, da sie zumeist einen gewissen Interpretationsspielraum bieten.

Logische Szenarien sind detaillierte Beschreibungen aller Szenarioinhalte. D. h., dass im Gegensatz zu funktionalen Szenarien alle statischen und dynamischen Elemente mit all ihren Zustandsvariablen abgebildet werden müssen. Zur Definition der Elemente und ihrer Variablen wird in der Regel eine formale Notation genutzt. Um die Brücke zwischen funktionalen und konkreten Szenarien zu schlagen, werden die Zustandsvariablen durch Wertebereiche und optionale Wahrscheinlichkeitsverteilungen über diesen angegeben. Somit repräsentiert ein logisches Szenario ebenfalls eine Menge an konkreten Szenarien und eignet sich u. a. dazu, vor der Systementwicklung situationsabhängige Grenzwerte festzulegen und Anforderungen an das umzusetzende System abzuleiten. [Menzel et al., 2018]

Konkrete Szenarien ähneln den logischen, erfordern aber für jede Zustandsvariable einen konkreten Wert statt eines Wertebereichs. Somit bildet ein konkretes Szenario genau einen Ausgangszustand ab. Aus einem logischen Szenario mit kontinuierlichen Wertebereichen können daher beliebig viele konkrete Szenarien abgeleitet werden, da die Schrittweite zur Konkretisierung der Werte beliebig klein gewählt werden kann. [Menzel et al., 2018]

Tabelle 6: Eigenschaften und Beispiele funktionaler, logischer und konkreter Szenariospezifikationen

	Funktional	**Logisch**	**Konkret**
Inhalt	Offener Szenarioraum	Geschlossener Szenarioraum	Eindeutiges Szenario
Art nach [Menzel, Bagschik und Maurer, 2018]	Darstellung auf semantischer Ebene durch natürlichsprachliche Formulierung und domänenspezifische Terminologie. Der Detailgrad ist anwendungsfallabhängig.	Darstellung auf Zustandsraumebene durch eine formale Notation. Die Abbildung der Entitäten und deren Beziehungen erfolgt durch die Angabe von Wertebereichen.	Eindeutige Darstellung auf Zustandsraumebene. Abbildung der Entitäten und deren Beziehungen erfolgt durch die Angabe konkreter Werter für jeden Parameter.
Abstraktion	Hoch	Mittel	Niedrig
Anzahl	Gering	Mittel	Groß
Beispiel (gekürzt)	*„Zwei Containerschiffe befinden sich auf offener See mit moderater Geschwindigkeit auf einem potentiellem Kollisionskurs.“*	Containerschiff 1: Länge = 400 m Kurs = [80,…,100]° Gesch. = [15,…,20] kn Containerschiff 2: Länge = 400 m Kurs = [260,…,280]° Gesch. = [15,…,20] kn	Containerschiff 1: Länge = 400 m Kurs = 89° Gesch. = 18 kn Containerschiff 2: Länge = 400 m Kurs = 271° Gesch. = 16 kn

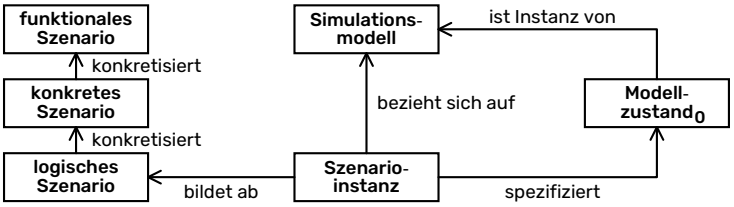


Abbildung 37: Einordnung des vorgeschlagenen Begriffs einer Szenarioinstanz

Die wichtigsten Eigenschaften der drei Szenariospezifikationstypen werden in der Tabelle 6 vergleichend dargestellt und jeweils durch ein Minimalbeispiel veranschaulicht. Allen drei Typen ist gemein, dass eine direkte Verwendung als Eingabe für eine Simulationsanwendung kein Kriterium darstellt. Vielmehr liegt der Fokus auf der Strukturierung der Herleitung konkreter Szenarien aus Experten-Formulierungen (funktionale Szenarien), der Eindeutigkeit durch Verwendung formaler Notationen und der Katalogisierbarkeit. Sollen simulative Testläufe der spezifizierten Szenarien durchgeführt werden, müssen die Inhalte in ein maschinenlesbares Format überführt werden, welches die Fähigkeiten, Elemente und Beziehungen des Simulationsmodells möglichst genau abbildet (vgl. Abbildung 36). Im Rahmen dieser Arbeit wird der Begriff *Szenarioinstanz* vorgeschlagen und verwendet, um diesen vierten, den konkreten Szenarien nachgelagerten Szenariospezifikationstyp zu bezeichnen. Eine *Szenarioinstanz* ist also die *persistente deskriptive Darstellung der Szenarioinhalte mit Bezug zum Simulationsmodell* (vgl. Abbildung 37). Eine Szenarioinstanz muss dabei maschinenlesbar und -interpretierbar sein, um als Eingabe für eine Simulationsanwendung und für die Instanziierung des Ausgangszustands eines Simulationslaufs (Modellzustand$_0$) auf Basis des Simulationsmodells genutzt werden zu können.

Neben einer klassischen Konfiguration über grafische Benutzeroberflächen innerhalb der Simulationsanwendung selbst haben sich für die Auswahl der an einem geplanten Simulationslauf teilnehmenden Elemente des Gesamtmodells und die Parametrierung dieser insbesondere *Szenariospezifikationssprachen* (engl.: Scenario Description Language, SDL) etabliert. SDLs stellen eine Untergruppe der Gruppe der *domänenspezifischen Sprachen* (engl.: Domain-Specific Language, DSL) dar, welche bereits seit Langem darauf abzielt, die Kommunikation zwischen Programmierern und Domänen-Experten zu erleichtern [Fowler, 2010]. Die Speerspitze in Bezug auf die Anzahl existierender SDLs stellt abermals der Bereich der Straßenverkehrssimulationen dar. SDLs können dabei unterschiedlicher Ausprägung sein: Einige sind offen und frei verfügbar, andere geschlossen und kommerziell, manche textbasiert, einige grafisch. Zwei Beispiele sind der etablierte offene Standard *OpenSCENARIO*[43] der *Association for Standardization of Automation and Measuring Systems* (ASAM), welcher darauf abzielt, den Inhalt eines Simulationslaufs textbasiert zu beschreiben, und am anderen Ende des Ausprägungsspektrums die visuelle Beschreibungssprache *Traffic Sequence Charts* (TSC), die eine grafische Darstellung von Trajektorienfamilien basierend auf einer formalen Semantik ermöglicht. Ein ausführlicherer Überblick über eine Auswahl der etabliertesten SDLs im Automobilbereich wird in [Ma et al., 2021] gegeben.

Wissenschaftliche Bemühungen bzgl. solcher Sprachen und Methoden zur Spezifikation von Szenarien als Basis für die Durchführung simulativer Testläufe gibt es schon lange. So beschrieben u. a. Kearney et al. [1999] bereits vor der Jahrtausendwende eine Sprache, um die von ihnen als Szenario-Programme bezeichneten Szenariobeschreibungen zu erstellen. Eine systematische Übersichtsstudie

[43] `https://www.asam.net/standards/detail/openscenario-xml/` [letzter Zugriff: 17.02.26]

aus dem Jahr 2020 kommt trotzdem zu dem Schluss, dass in dem Großteil der wissenschaftlichen Veröffentlichungen, die neue Algorithmen zur Navigation und Kollisionsvermeidung für automatisierte Schiffe vorgeschlagen haben, zwar meist simulative Testläufe zu Evaluierungszwecken durchgeführt wurden, ein Teil davon aber nicht szenariobasiert erfolgte und in den übrigen Fällen nur sehr wenige einfache und manuell erstellte Szenarien getestet wurden. [Porres et al., 2020b].

Ein mit Hilfe einer SDL präzise spezifiziertes Szenario allein ist von geringem Nutzen. Um dieses für simulative Testläufe nutzen zu können, muss zusätzlich beachtet werden, welche Simulationsanwendungen welche SDLs importieren, interpretieren und zur Initialisierung eines Simulationslaufs nutzen können. Für viele aktuelle Simulationsprogramme trifft das lediglich auf ein oder zwei SDLs zu. Andere ermöglichen wiederum ausschließlich die Konfiguration über eigene geschlossene proprietäre Formate oder über eine spezielle grafische Benutzeroberfläche innerhalb der Simulationsanwendung. Letztlich bedeutet das, dass in einem solchen Fall die Wahl der SDL die Wahl der Simulationsanwendung bestimmen kann und umgekehrt [Ma et al., 2021]. Viele der etablierten SDLs sind stark an ein bestimmtes Simulationsmodell und somit an die entsprechende Simulationsanwendung gebunden. Diejenigen, die unabhängig von einem bestimmten Simulationsmodell definiert sind, können zwar theoretisch flexibel eingesetzt werden, doch ihr Einsatz führt in der Regel zu der Notwendigkeit von Anpassungen der Simulationsanwendung, damit die mit der SDL beschriebenen Inhalte des Szenarios durch diese interpretiert und in den initialen Simulationszustand überführt werden können.

Werden die Elemente und Beziehungen einer klassischen Simulationsanwendung, wie sie in Abbildung 35 dargestellt sind, um eine solche von der Simulationsanwendung und dem Simulationsmodell prinzipiell unabhängige Art der Spezifikation einer Szenarioinstanz erweitert, lassen sich die in Abbildung 38 dargestellten Beziehungen erkennen. Die linke Seite der Abbildung veranschaulicht den strukturellen Aufbau einer Szenarioinstanz, während die rechte Seite den Aufbau einer Simulationsanwendung bzw. die Inhalte eines Simulationslaufs und des zugrunde liegenden Modells illustriert. Das Erstellen einer Szenarioinstanz umfasst die Auswahl und Parametrisierung von Elementen und Beziehungen, die durch das abstrakte Modell der genutzten SDL zur Verfügung gestellt werden. Im maritimen Kontext könnte die Menge der verfügbaren Elemente unter anderem verschiedene Arten von Schiffen verschiedener Typen, Seezeichen und etwaige einschränkende Beziehungen zwischen diesen umfassen. Ein konkretes Szenario zum Testen eines Assistenzsystems könnte dann aus zwei Schiffen des Typs *Containerschiff* bestehen, die sich auf Kollisionskurs befinden und deren Bahnen durch Seezeichen eingeschränkt sind. Analog zum Begriff des *Simulationsmodells* wird in dieser Arbeit für dieses abstrakte Modell und die daraus hervorgehenden Instanzen der Begriff *Szenariomodell* ge-

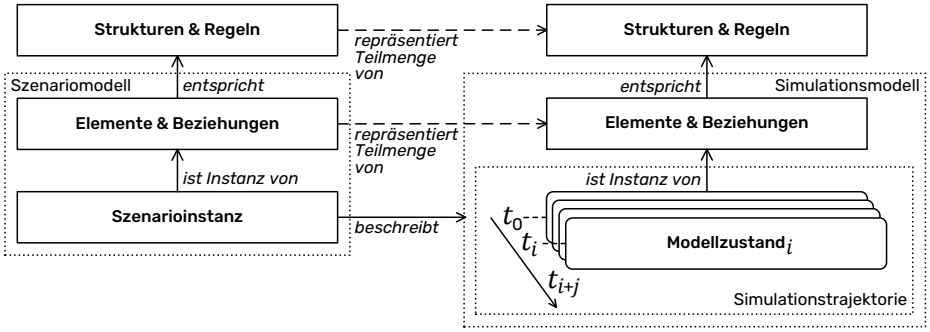


Abbildung 38: Beziehungen zwischen den Modellen und ihrer Bestandteile, die für die Durchführung einer szenariobasierten Simulation notwendig sind.

nutzt. Die Struktur und die Eigenschaften der verfügbaren Elemente sind dabei ebenfalls nicht vollständig willkürlich, sondern entsprechen einer, allen Elementen gemeinsamen Grundlage von vorgegebenen Strukturen und Regeln. Es ergibt sich so auf beiden Seiten der Abbildung ein nahezu deckungsgleiches Bild: Auch auf der Seite des Simulationsmodells bilden Vorgaben die Grundlage für die Definition einer Reihe von Elementen und Beziehungen, die Teil eines Simulationslaufs sein können (vgl. Abschnitt 7.4.1). Schlussendlich nimmt jedoch nur eine durch die Szenarioinstanz spezifizierte Teilmenge der möglichen Elemente an einem konkreten Simulationslauf teil.

Damit ein Szenario eine gültige Beschreibung des Ausgangszustands eines Simulationslaufs darstellt, ist es erforderlich, dass die Inhalte der Schichten auf der linken Seite eine Teilmenge der Inhalte derer auf der rechten Seite mit größtmöglicher Genauigkeit repräsentieren. Für den Fall, dass die Szenarioinstanz Elemente oder Strukturen enthält, die der Simulation nicht bekannt sind, ist es nicht möglich, das Szenario vollständig in einem Simulationslauf abzubilden: Es kommt zum Informationsverlust im Rahmen der Übertragung der Szenarioinhalte in eine Simulationstrajektorie. Umgekehrt kann das Potenzial der Simulationsanwendung und ihres -modells nicht vollständig ausgenutzt werden, wenn bestimmte Elemente, die die Simulationsanwendung darstellen kann, nicht vollständig im Szenario abgebildet werden können, weil die strukturellen Vorgaben für das spezifische Szenarioformat dies nicht hergeben oder die Menge der modellierten Elemente auf der Seite der Simulationsanwendung größer ist als die des zugrunde liegenden Szenariomodells. Letzteres wäre z. B. der Fall, wenn die Simulation in der Lage ist, Schiffe mit einer Anfangsgeschwindigkeit zu Beginn der Simulation darzustellen, das Element *Schiff* im Szenariomodell aber keinen Parameter für die Anfangsgeschwindigkeit bietet. Diese Abhängigkeit und die daraus resultierenden Herausforderungen für Anwender werden auch auf der Webseite des ASAM Standards OpenSCENARIO in der Version 2.0.0 [44] prominent erwähnt: „*Neither version of ASAM OpenSCENARIO guarantees exact reproducibility across different tools due to the high amount of implementation-specific factors that may influence the simulation results.*“ Auch die Spezifikation des Standards OpenSCENARIO XML in der Version 1.3.0 [45] stellt einleitend im Abschnitt *6.1 Architecture* dar, dass die Simulationsanwendung an den Standard angepasst werden muss, damit die Szenarien von dieser ausgeführt werden können. Dafür müssen hauptsächlich Modelle implementiert werden, die denen des Standards entsprechen, und Interfaces zur Verfügung gestellt werden, die von der *Director* genannten Komponente angesprochen werden können.

7.4.3 METAMODELLE & MODELLING-SPACES

Jede Simulation dynamischen Verhaltens setzt zunächst die Erstellung eines Modells voraus, das eben dieses Verhalten beschreibt [Mattern, 1996] (vgl. auch Abschnitt 8.3.1). Die in den vorherigen Abschnitten besprochenen Elemente und Beziehungen der Simulations- und Szenariomodelle entsprechen dabei im weiteren Sinne der geläufigen Definition eines Datenmodells: „Ein Datenmodell […] bestimmt die grundsätzlichen Strukturen, die Beziehungen, die prinzipiell möglich sind, und die Eigenschaften, die zugeordnet werden können. Im Modellierungsprozess werden für die fachlich notwendigen Erscheinungen der Wirklichkeit im Rahmen der grundlegenden Vorgaben des Datenmodells Festlegungen getroffen.“ [46] [Bill, 2016]

[44] https://www.asam.net/standards/detail/openscenario/v200/ [letzter Zugriff: 17.02.26]

[45] https://www.asam.net/index.php?eID=dumpFile&t=f&f=4092&token=d3b6a55e911b22179e3c0895fe2caae8f5492467 [letzter Zugriff: 17.02.26]

[46] Auslassung;

Nach [Ferstl & Sinz, 2013] legt ein Metamodell das Begriffssystem fest, das zur Spezifikation von Modellen genutzt werden kann: nutzbare Modellierungselemente, Beziehungen zwischen diesen, Regeln für die Verknüpfung von den Elementen anhand von Beziehungen sowie die genaue Bedeutung dieser Konstrukte. Die im vorangegangenen Abschnitt identifizierten und vereinfacht als *Strukturen & Regeln* in Abbildung 36 und Abbildung 38 dargestellten externen oder selbstauferlegten Vorgaben können nach dieser Definition als Metamodell dieser Datenmodelle betrachtet werden. Der Begriff *meta* hat seinen Ursprung in der griechischen Sprache und bedeutet *über*. Vereinfacht ausgedrückt sind Metamodelle also Modelle über Modelle: Für ein Metamodell ist die zu beschreibende Wirklichkeit (vgl. Definition des Begriffs *Modell* in Abschnitt 7.3.1) selbst ein Modell. Das Metamodell beschreibt die Struktur des Modells und steht somit in einer Klasse-Instanz-Beziehung mit diesem. Jede formale Sprache, wie Java, XML, SQL oder UML, besitzt ein Metamodell: Ein XML-Schema stellt das Metamodell zu einem, entsprechend diesem Schema gültigen, XML-Dokument dar. Die BNF (Backus Naur Form) ist eine Metasprache für die Beschreibung der Syntax von Programmiersprachen, d. h., die BNF Beschreibung der Sprache Java ist eine Instanz der BNF und eine in BNF vorliegende Beschreibung der Sprache Java ist ein Metamodell der Sprache Java (vgl. [Vangheluwe & Lara, 2002]). Ein konkretes Beispiel für eine solche Modellierungshierarchie kann anhand einiger öffentlich zugänglicher Artefakte des OpenSCENARIO-Standards gegeben werden:

Metamodell: Informationen über den vorgeschriebenen Aufbau, die zu beachtenden Einschränkungen sowie die möglichen inhaltlichen Strukturen und Inhalte eines durch OpenSCENARIO beschriebenen Szenarios werden von ASAM unter anderem in Form einer XML Schema Definition (XSD) zur Verfügung gestellt[47].

Modell: Ein XML-Dokument, das inhaltlich und strukturell der OpenSCENARIO Schema Definition entspricht, beschreibt den Ausgangszustand und den Verlauf eines konkreten Verkehrssimulationslaufs und ist daher eine abstrakte Beschreibung (das Modell) dessen. Einige Beispiele für solche Szenarien werden durch ASAM als Anhang der Online-Dokumentation zum Download bereitgestellt[48].

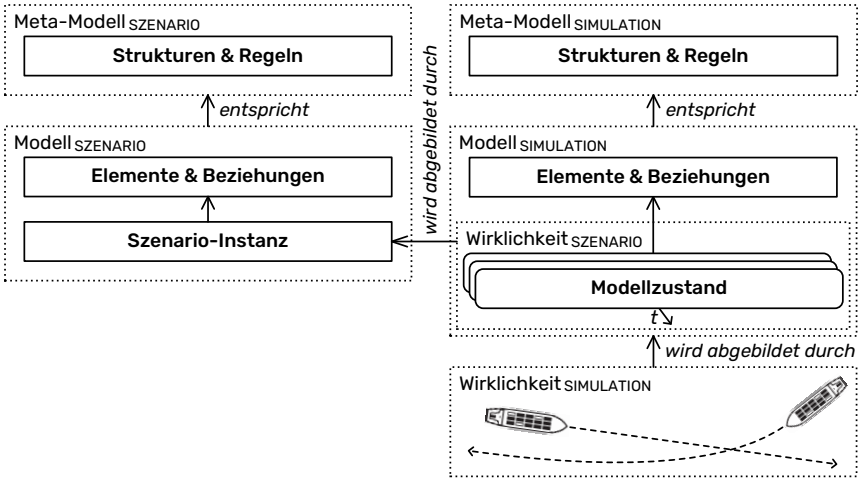


Abbildung 39: Orthogonale Modellierungshierarchien (engl.: Modelling Spaces) einer klassischen szenariobasierten Verkehrssimulation mit getrennten, eigenständigen Szenario- und Simulationsmodellen.

[47] https://publications.pages.asam.net/standards/ASAM_OpenSCENARIO/ASAM_OpenSCENARIO_XML/latest/_attachments/generated/ASAM_OpenSCENARIO_v1.3.1_Schema.zip [letzter Zugriff: 17.02.26]

[48] https://publications.pages.asam.net/standards/ASAM_OpenSCENARIO/ASAM_OpenSCENARIO_XML/latest/_attachments/generated/ASAM_OpenSCENARIO_v1.3.1_Examples.zip [letzter Zugriff: 17.02.26]

Wirklichkeit: Die beschriebene Wirklichkeit ist in diesem Fall nicht die Realität in Form einer realen Testfahrt, sondern der Ausgangszustand und der Verlauf des durch das Szenario beschriebenen Simulationslaufs (vgl. Abbildung 39).

Durch Einordnung der zuvor identifizierten und in Abbildung 38 dargestellten Modell-Bestandteile szenariobasierter Simulationsläufe in eine solche Metamodell Hierarchie ergibt sich das in Abbildung 39 gezeigte Bild. Deutlich zu erkennen sind die zwei separaten Modellierungshierarchien des Szenariomodells und des Simulationsmodells, welche nach Gasevic et al. zwei unterschiedliche sogenannte *Modelling Spaces* darstellen. Da die Wirklichkeit des Spaces SZENARIO das Modell des Spaces SIMULATION ist und somit der eine Space Konzepte des anderen Spaces modelliert, handelt es sich in diesem Fall konkret um *orthogonale* Modelling Spaces.

8 SIMULATIONSANWENDUNGEN ANDERER VERKEHRSZWEIGE

Mit dem bis hierhin aufgearbeiteten fachlichen und technischen Kenntnisstand sollen nun einige wenige beispielhafte Simulationsanwendungen und technische Szenariobeschreibungsansätze aufgezeigt werden, die in anderen Verkehrszweigen Einsatz finden. Die nachfolgenden Darstellungen sind auf die jeweils prominentesten Beispiele reduziert, da hier lediglich ein Überblick gegeben werden soll, welche Lösungsansätze in anderen Domänen verfolgt werden und welchen Herausforderungen diese ggf. zusätzlich oder ebenfalls gegenüberstehen.

In Hinblick auf den Automobilbereich sei zunächst auf zwei umfangreiche, aktuelle Übersichtsarbeiten verwiesen: Zhang et al. [2024] bieten einen vergleichenden Überblick über Simulationsanwendungen und entsprechende Datensätze. Da Bruto Costa et al. [2025] haben auf Basis einer systematischen Literaturanalyse einen umfangreichen Überblick über aktuelle Best Practices und Forschungsaktivitäten in Bezug auf den szenariobasierten Sicherheitsnachweis automatisierter Straßenfahrzeuge erarbeitet. Eine Simulationsanwendung, die weitverbreitete Anwendung findet, ist *Car Learning to Act* (CARLA) [Dosovitskiy et al., 2017]. CARLA ist ein 3D-Fahrsimulationssystem für Straßenfahrzeuge, das als Open-Source-Schicht auf der *Unreal Engine* aufsetzt. Es zielt auf die Unterstützung von Ausbildung, Entwicklung und Validierung von Modellen und Fahrfunktionen für autonomes Fahren in städtischem Verkehr. Städtische Umgebungen sowie 3D-Modelle statischer und dynamischer Objekte wie Fahrzeuge, Gebäude und Fußgänger sind kostenlos enthalten. Einer der genannten Gründe für die Entwicklung war, dass bestehende Lösungen entweder nicht detailliert genug waren oder keine detaillierte Bewertung von Fahrstrategien und -funktionen unterstützten. CARLA unterstützt eine flexible Einrichtung von virtuellen Sensoren und eine einfache Python-API [Wu et al., 2022]. Mit Hilfe dieser API geschriebene Client-Skripte können u. a. einen Akteur erstellen, diesem Sensoren zuordnen und anschließend die Sensordaten abrufen, verarbeiten und die vom Controller benötigten Parameter berechnen. Die berechneten Werte werden dann an den CARLA-Simulator zurückgesendet. Ein dediziertes Szenariomodell im Sinne der Definitionen dieser Arbeit existiert somit nicht. Eine einfache Integration externer, zu testender Systemkomponenten über gesamtheitliche Fahrfunktionen hinaus wird zudem durch die begrenzte Anzahl an möglichen Steuerbefehlen erschwert.

Weitere erwähnenswerte Simulationsanwendungen für automobilen Verkehr in unspezifischer Reihenfolge sind *Paracosm* [Majumdar et al., 2021], *LGSVL Simulator* [Rong et al., 2020], *OpenDS* [Math et al., 2013] und *The Open Racing Car Simulator* (TORCS) [Wymann et al., 2015].

Im Bereich der Szenariospezifikation hat der bereits erwähnte Standard *OpenSCENARIO* [Fan & Krishnan, 2020] weite Verbreitung und Akzeptanz erlangen können. Dieser steht in enger Verbindung mit den zwei Standards *OpenDRIVE* und *OpenCRG*. Diese ergänzen einander und decken in ihrer Gesamtheit den statischen und den dynamischen Inhalt eines Straßenverkehrsszenarios ab. Szenariospezifikationen liegen hier in der Regel in Form von *XML* vor. Zur Abbildung von Dynamik werden Manöver explizit beschrieben und mit Zustandsbedingungen verknüpft. Viele der etablierten Simulationsanwendungen unterstützen Dateien im OpenSCENARIO-Format als Eingabe zum Starten von Simulationsläufen. OpenSCENARIO ist dabei vollständig auf den Einsatz in der Automobildomäne ausgerichtet und forciert eine vollständige Trennung des zugrunde liegenden Modells von der ausführenden Simulationsanwendung und ihrem internen Modell. Die Einbindung von zu testenden externen Systemen und Komponenten ist derweil nicht explizit angedacht.

AirSim [Shah et al., 2018] ist eine offene Simulationsanwendung, die auf der Unreal Engine basiert. Das Ziel von AirSim ist es, zum einen physikalisch und zum anderen auch visuell realistische Simulationsläufe zu bieten. So sollen sowohl Entwicklungs- und Testaktivitäten unterstützt werden als auch eine virtuelle Alternative für reale Trainingsdaten für das Training von auf Machine Learning basierenden Ansätzen geboten werden. Primär adressiert werden dabei kleinere unbemannte Luftfahrzeuge. Die Simulationsanwendung umfasst eine Physik-Engine, die *Echtzeit-Hardware-in-the-Loop-Simulationen* unterstützt. AirSim stellt sich dabei den Anspruch, einfach erweiterbar zu sein, um Erweiterungen für z. B. neue Fahrzeugtypen zu ermöglichen. Dass dieses Ziel erreicht wurde, konnte u. a. dadurch gezeigt werden, dass in [Yao et al., 2018] automobile Dynamikmodelle, Sensormodelle und Umgebungsmodelle umgesetzt und angebunden wurden. AirSim setzt sich aus den Komponenten *Umgebungsmodell*, *Fahrzeugmodelle*, *Sensormodelle*, *Rendering Schnittstelle*, *öffentliche Programmierschnittstelle* sowie einer Schnittstelle zur direkten Anbindung der Firmware des Fahrzeugs zusammen. Fahrzeugmodelle beschreiben Fahrzeuge als eine Kombination aus starren Körpern und Aktuatoren, die gerichtete Kräfte und Drehmomente erzeugen können. Das integrierte physikalische Modell umfasst Phänomene wie die Erdanziehungskraft, Luftdichte, Luftdruck und magnetische Felder, wobei gezielt Modelle gewählt wurden, die einen akzeptablen Kompromiss zwischen Genauigkeit und Echtzeitfähigkeit bieten. Funktionalitäten wie die Darstellung der Umgebung, die Berechnung von Kollisionen und die Spezifikation von Szenarien delegiert AirSim zum überwiegenden Teil an die eingesetzte Unreal Engine. Somit setzt die Szenariospezifikation die Einarbeitung und Nutzung der entsprechenden Entwicklungsumgebung voraus und gestaltet sich, auch aufgrund einer steilen Lernkurve, als zeitaufwendig und komplex.

Das offene *Trick* Simulation Toolkit [Penn & Lin, 2016] der US-Bundesbehörde *National Aeronautics and Space Administration* (NASA) existiert bereits seit den 1980er Jahren, wurde seitdem stets weiterentwickelt und wird durch die NASA regelmäßig für die Umsetzung hochauflösender Simulationen eingesetzt. Das Ziel von Trick ist es, Simulationsentwickler in einer Art zu unterstützen, die es ihnen ermöglicht, sich auf ihren Fachbereich zu konzentrieren, statt wiederkehrende, zeitaufwendige Architekturprobleme zu lösen und erforderliche Funktionen wiederholt zu implementieren. Um trotz des gebotenen hohen Grades an Wiederverwendbarkeit und Standardisierung flexibel einsetzbar zu sein, setzt Trick auf eine Vorabbeschreibung der Variablen, Daten, Typen, Funktionen und Ablaufplanung durch den Entwickler in Form eines *Simulation Definition Files*. Die zentrale Komponente eines Simu-

lationslaufs ist der Scheduler, der Simulationszeit und -modi verwaltet sowie durch *Jobs* beschriebene Modellfunktionen in einer definierten Reihenfolge ausführt. Durch die gebotene hohe Auflösung, die Fähigkeit zur Skalierung, die Echtzeitfähigkeit und die deterministische Ausführungsreihenfolge eignet sich Trick gut für die Umsetzung akkurater Physiksimulationen und die Durchführung großangelegter Parameterstudien. Agentenbasierte und wirkstrukturabbildende Simulationen sind nicht direkt vorgesehen und daher nicht ohne Weiteres umsetzbar. Zudem bietet der *Variable Server* lediglich eingeschränkte Möglichkeiten zur Interaktion mit externen Anwendungen.

9 ANFORDERUNGEN AN MARITIME SIMULATIONSANWENDUNGEN

Auf Basis der vorangegangenen Untersuchung der zentralen Aspekte einer szenario- und simulationsbasierten Verifikation und Validierung moderner automatisierter maritimer Fahrsysteme können an dieser Stelle bereits einige wesentliche Anforderungen formuliert werden. Aus den Inhalten der vorangegangenen Abschnitte kann geschlussfolgert werden, dass die Erfüllung dieser entscheidend ist für die effektive Umsetzung eines entwicklungsintegrierten V&V-Prozesses während der Neuentwicklung und Einführung entsprechender maritimer softwarebasierter Systeme.

Da sich gezeigt hat, dass eine getrennte Betrachtung nötig ist, um sowohl begriffliche als auch inhaltliche Klarheit zu schaffen, sind die ermittelten Anforderungen in zwei Kategorien eingeteilt: (A1) Anforderungen an die Simulationsanwendung und (A2) Anforderungen an den Szenariomodellierungsprozess. Die Anforderungen werden nachfolgend kurz beschrieben und die Notwendigkeit der Erfüllung wird mit Rückbezug auf die Abschnitte 6 und 7 begründet. Im Fokus steht die Bedeutung für den Einsatz innerhalb der V&V-Prozesse während der Entwicklung maritimer softwarebasierter Fahrsysteme.

Anforderungen an die Simulationsanwendung (A1)

A1.1 – Simulation maritimen Verkehrs

Die Simulationsanwendung muss in der Lage sein, den zeitlichen Ablauf von maritimen Szenarien unterschiedlichen Detailgrades zu simulieren. Dies erscheint auf den ersten Blick selbstverständlich, jedoch gibt es auch Multi-Domänen-Simulationsanwendungen, die den maritimen Bereich nicht ausreichend für den vorgesehenen Zweck abbilden, und solche, die in dem abgebildeten Detailgrad nicht oder nur wenig flexibel sind.

A1.2 – Interoperabilität

Es muss möglich sein, gängige Ausprägungen eines zu testenden Systems (engl.: System Under Test, SUT) und seiner möglichen Komponenten in allen Phasen des Systems-Engineering-Lebenszyklus am Simulationslauf teilnehmen zu lassen. Das heißt konkreter, dass die Kommunikation zwischen externen Modellen (Model-in-the-Loop, MiL), Software-Funktionseinheiten (Software-in-the-Loop, SiL), Hardwarekomponenten und -systemen (Hardware-in-the-Loop, HiL) sowie integrierten Schiffen (Vehicle-in-the-Loop, ViL) [Brinkmann & Hahn, 2017; Pfeffer & Leichsenring, 2016] und der Simulation ohne Anpassung der Implementierung der Simulationsanwendung selbst möglich sein muss.

A1.3 – Verschiedene Abstraktionsebenen

Um in jeder Phase des Entwicklungsprozesses nützlich zu sein, muss die Verkehrssimulation in ihrem Abstraktions- und Detailgrad variabel sein. So kann es z. B. in der Anfangsphase der Entwicklung – der Modellbildung – ausreichend sein, monolithisch modellierte Schiffe zu simulieren, deren Verhal-

ten nur auf einer einzelnen einfachen Fahrfunktion beruht. Im weiteren Verlauf der Entwicklung nach dem V-Modell wird es dagegen notwendig sein, ein Schiff und seine Komponenten als System-of-Systems (SoS) zu modellieren. Ersteres hilft, den Testaufwand gering zu halten, wobei Letzteres die für eine erfolgreiche testbasierte Abnahme notwendige detaillierte Analyse aller wichtigen Teilsysteme und Komponenten ermöglicht. Entsprechend der Anforderung *A1.2 Interoperabilität* wird damit auch die Voraussetzung geschaffen, Teilsysteme (z. B. Alarmierungssysteme wie das *Maritime Traffic Alert and Collision Avoidance System* (MTCAS) [Steidel & Hahn, 2019]) als SUT in die Simulationsläufe einzubinden.

A1.4 – Beobachtbarkeit

Um simulative Testläufe zur Laufzeit maschinell auswerten zu können, muss es möglich sein, die Simulation, ihre Inhalte und deren Parameter zu jedem simulierten Zeitschritt beobachten zu können. „Beobachten" meint an dieser Stelle nicht das Nachverfolgen des Simulationslaufs durch einen menschlichen Nutzer anhand einer grafischen Visualisierung, sondern vielmehr die Möglichkeit, präzise Werte zu allen Parametern ausgegeben zu bekommen, um diese für eine Bewertung nutzen zu können. Die Bewertung des Testlaufes kann dabei zum Beispiel durch den Vergleich der beobachteten Werte mit als validen vordefinierten Wertebereichen geschehen [Hake et al., 2024]. Es muss somit möglich sein, einfache Werte und Parameter, wie die Position eines Schiffes oder die aktuelle Drehzahl des Motors, auszulesen. Idealerweise ist es auch möglich, den zeitlichen Verlauf der gewünschten Werte persistieren zu lassen, um die Auswertung im Nachhinein durchführen zu können oder um die Daten mit externen Tools zu bearbeiten und zu vergleichen.

A1.5 – Startzustand und Parametrisierung

Um einen zielgerichteten Simulationslauf durchführen zu können, muss ein vordefinierter Startzustand konfiguriert werden können, in welchem alle Simulationsinhalte mit ihren jeweiligen Eigenschaften zum Zeitpunkt t_0 definiert sind. Zusätzlich müssen den aktiven Verkehrsteilnehmern Ziele und Werte vorgegeben werden können, die das simulierte Verhalten dieser Teilnehmer zur Laufzeit bestimmen. Dies entspricht der von Ulbrich et al. [2015] formulierten Definition des Begriffes *Szenario*. Die Simulationsanwendung muss also in der Lage sein, ein Szenario einzulesen und dieses als Ausgangsparametrierung zu verwenden, um die Simulationsinhalte für den Zeitpunkt t_0 zu initialisieren. Zusätzlich ist es von Interesse, welche Arten von Szenariomodellen und -formaten als Input verarbeitet werden können, da sich eine persistente Speicherung und Übertragung der Szenarien sowie eine hohe Nutzbarkeit positiv auf die Vergleichbarkeit von Ergebnissen auswirken (vgl. *A2.2*, *A2.3*).

A1.6 – Erweiterbarkeit und Anpassbarkeit

Die Menge der passiven und aktiven Verkehrskomponenten, die Teil eines Simulationslaufs sein können, sollte leicht erweiterbar und anpassbar sein. Muss etwa ein neuer Schiffstyp getestet werden, der zum Zeitpunkt der Programmierung der Simulationsanwendung noch nicht existierte und daher nicht Bestandteil dessen internen Modells ist, muss dieser Typ in die Simulation integriert werden können, damit die Durchführung eines entsprechend geeigneten Simulationslaufs möglich ist. Die Simulation muss also leicht an die Bedürfnisse der jeweiligen Tests durch Erweiterungen oder Änderungen am Simulationsmodell angepasst werden können. Dies hängt bis zu einem gewissen Grad auch mit der Forderung nach unterschiedlichen Abstraktionsgraden *A1.2* zusammen.

Anforderungen an den Szenariomodellierungsprozess (A2)

A2.1 – Abbildung maritimer Szenarien

Analog zur Anforderung *A1.1* muss es möglich sein, durch Instanzen des Szenariomodells maritime Verkehrsszenarien abbilden zu können. Die Abbildung muss geeignet sein, um diese im Rahmen der simulativen Verifikation und Validierung (V&V) von maritimen Fahrsystemen verwenden zu können.

A2.2 – Austauschbarkeit

Um die Arbeit mit Simulationsanwendungen und die Katalogisierung von Szenarien zu ermöglichen, muss ein Szenariomodell austauschbar sein. Eine reine In-Memory- oder In-Application-Haltung des Modells und der individuellen Szenarioinstanzen ist nicht ausreichend. Dies eröffnet u. a. die Möglichkeit, Szenarioinstanzen lokal auf einem Arbeitsplatzcomputer zu erstellen und diese dann z. B. auf einem leistungsfähigeren Remote Server auszuführen. Zusätzlich werden die Nachvollziehbarkeit und Vergleichbarkeit von Ergebnissen gefördert.

A2.3 – Einheitliche Basis

Um breite Anwendung zu finden, sollten Szenariomodelle auf einer abstrakten, einheitlichen Basis aufgebaut werden können. Wenn das Szenariomodell oder -format von Grund auf für eine einzige spezifische Simulationsanwendung entwickelt wurde, besteht die Gefahr, dass es nur mit diesem System verwendet werden kann. Solche Single-Use Modelle sind zwar häufig sehr mächtig, sind aber ein Hindernis für die Vergleichbarkeit von Testergebnissen, den Austausch von Szenarien zwischen verschiedenen Entwicklungs- und Forschungsteams und damit auch für die zukünftige schnelle Weiterentwicklung des gesamten Forschungsfeldes.

A2.4 – Erweiterbarkeit und Anpassbarkeit

Da ein Szenariomodell die Inhalte des Simulationslaufs abbildet, muss es analog zu *A1.6* auch entsprechend erweiterbar und anpassbar sein. Wenn ein Szenario nicht alle Fähigkeiten der verwendeten Simulationsanwendung abbildet, werden Potenziale nicht ausgeschöpft. Kann das Szenario mehr abbilden als die Simulationsanwendung, besteht die Gefahr des Informationsverlustes bei der Initialisierung des Simulationsmodells aus dem gegebenen Szenariomodell.

A2.5 – Maschinenlesbarkeit

Das Format, in dem das Szenario als eine konkrete Modellinstanz persistiert wird, muss maschinenlesbar sein, damit die Simulationsanwendung es als direkten Input verwenden kann (vgl. *A1.5*).

A2.6 – Menschenlesbarkeit & Werkzeugunterstützung

Das Format, in dem das Szenario persistiert wird, muss entweder menschenlesbar und -erstellbar sein oder es muss eine ausreichende Werkzeugunterstützung für eine verständliche Darstellung und Erstellung vorhanden sein. Ist beides nicht der Fall, wird der V&V-Prozess erheblich erschwert, da die zu testenden Verkehrsszenarien zunächst unter Verwendung des Szenariomodells in Szenarioinstanzen überführt werden müssen. Dies geschieht in der Regel von Hand. Auch automatisierte intelligente Methoden, die Szenarien automatisch erstellen können, benötigen in der Regel eine Startparametrierung, von welcher aus der Zustandsraum exploriert wird (vgl. z. B. [Weber et al., 2019]).

A2.7 – Konfiguration der Beobachtung

In Bezug auf das Kriterium *A1.4* sollte die Konfiguration der von der Simulationsanwendung auszugebenden Parameter der Objekte eines Simulationslaufs bereits Teil des Szenariomodells sein. Ähnlich zu *A1.2* ergibt sich diese Anforderung aus der Notwendigkeit, individuelle Anpassungen der Simulations-

anwendung selbst für verschiedene Tests und zu testende Komponenten zu vermeiden. Dies würde ebenso wie die Anforderung *A1.5* zu einer vereinfachten Automatisierung von konsekutiven Durchführungen mehrerer szenariobasierter Tests beitragen.

10 AKTUELLER EINSATZ MARITIMER SIMULATIONEN ZU TESTZWECKEN

Die formulierten Anforderungen, die es durch eine Simulationsanwendung sowie den Szenariospezifikationsprozess zu erfüllen gilt (vgl. Abschnitt 9), damit szenariobasierte Simulationsläufe ein effektives und effizientes Werkzeug zur Unterstützung der Verifizierung und Validierung softwarebasierter maritimer Fahrassistenzsysteme darstellen, stellen das Ergebnis und somit den Abschluss der Erkundungsphase des ersten Teils dieser Arbeit dar (vgl. Abschnitt 5). Um die Frage nach dem Grad der Erfüllung der Anforderungen durch aktuell zum Einsatz kommende Simulationswerkzeuge zu beantworten, wurde im Rahmen der Untersuchungsphase ein systematisches Literaturreview durchgeführt. Dieses sollte diejenigen Simulationsanwendungen zutage fördern, die aktuell in wissenschaftlichen Arbeiten zu Testzwecken zum Einsatz kommen. Dabei wurde bewusst die Anwendersicht eingenommen: Von Interesse war, welche Werkzeuge diejenigen Forscher wählten, die mit der Notwendigkeit der Durchführung von simulativen Tests konfrontiert waren. Anschließend werden in Abschnitt 10.3 die identifizierten Simulationsanwendungen und -werkzeuge den in Abschnitt 9 formulierten Anforderungen abgleichend gegenübergestellt.

10.1 ERHEBUNGSMETHODE

Um die wesentliche Rolle eines systematischen Literaturreviews zu erfüllen, müssen bei der Planung und Durchführung dieser bestimmte Anforderungen erfüllt werden. Sowohl Kitchenham & Charters [2007] als auch vom Brocke et al. [2009] messen der Validität und Reliabilität dabei die höchsten Stellenwerte zu. Validität und Reliabilität werden durch eine sorgfältige, umfassende und rigorose Dokumentation des Suchprozesses erreicht, in der alle im Vorfeld getroffenen Entscheidungen, wie die Auswahl der Datenbanken, die verwendeten Schlüsselwörter, die zu erfassenden Zeiträume und die Auswahlkriterien, detailliert beschrieben werden [Webster & Watson, 2002; Cooper, 1988]. Eine derart detaillierte und vollständige Dokumentation gewährleistet Transparenz und Reproduzierbarkeit des Suchprozesses und dessen Ergebnisses, ist somit fest mit den Ergebnissen der Analyse verbunden und muss zusammen mit diesen dargestellt werden, wenn die Ergebnisse nicht an Wert und Aussagekraft verlieren sollen.

Daher werden das Vorgehen und seine zentralen Parameter im Folgenden detailliert beschrieben und begründet. Diese strukturierte Vorgehensweise wurde bereits im Vorfeld festgelegt, um eine gezielte und systematische Literatursuche und -auswertung frei von wertenden Einflüssen zu ermöglichen. Die folgenden vier Unterabschnitte sind chronologisch geordnet, um einen klaren Überblick über den Prozess zu geben: Zunächst werden die allgemeinen Merkmale der Literatursuche und -auswertung vorgestellt und begründet. Anschließend wird begründet dargestellt, wo und wie nach wissenschaftlicher Literatur gesucht wurde, einschließlich der Kriterien für die anschließende Filterung der Ergebnisse bezüglich ihrer Relevanz. Schließlich wird die integrative Analyse genauer erläutert, im Rahmen derer die gesammelten Daten synthetisiert und interpretiert wurden.

Durch die Einhaltung der genannten Kriterien stellen die in Abschnitt 10.3 gezeigten Ergebnisse eine zuverlässige Synthese des vorhandenen Wissens dar. Auf dieser Basis konnten neue Erkenntnisse in

Tabelle 7: Merkmale der durchgeführten Studie nach [Cooper, 1988].

Charakteristik	Ausprägung
Fokussierung	Forschungsergebnisse und Anwendungen
Zielsetzung	Integrativ (Hervorheben offener Forschungslücken)
Perspektive	Neutrale Repräsentation
Umfang	Selektiv
Ausrichtung	Methodisch
Zielgruppe	Spezialisierte sowie allgemeine Wissenschaftler

Form von konkreten Handlungsempfehlungen geschaffen werden, welche den angestrebten empirischen Charakter haben, und somit sowohl als Grundlage für den weiteren Verlauf der vorliegenden Arbeit als auch für zukünftige Forschung dienen.

10.1.1 CHARAKTERISTIKA

Um den Erfassungsbereich sowie das Ziel und den Zweck eines Literaturreviews a priori klar zu definieren, schlagen vom Brocke et al. [2009] vor, die etablierte Taxonomie für Literaturreviews von Cooper [1988] zu verwenden. Diese Taxonomie umfasst sechs Merkmale: Fokussierung (engl.: focus), Zielsetzung (engl.: goal), Perspektive (engl.: perspective), Umfang (engl.: coverage), Ausrichtung (engl.: organization) und Zielgruppe (engl.: audience). Die gewählten Ausprägungen dieser Merkmale sind in Tabelle 7 dargestellt und werden nachfolgend begründet.

Fokussierung: Die vorliegende Vorstudie konzentriert sich auf Forschungsergebnisse, die sich jeweils auf gleichartige Herausforderungen beziehen, und die konkreten Anwendungen dieser.

Zielsetzung: Die gefundenen Forschungsergebnisse wurden in einer integrativen Weise gruppiert und verglichen, um Forschungslücken zu ermitteln und Handlungsempfehlungen abzuleiten.

Perspektive: Um die Allgemeingültigkeit der abgeleiteten Erkenntnisse gewährleisten zu können, wurde während des gesamten Such- und Auswertungsprozesses eine neutrale Haltung gegenüber allen Arbeiten, Ergebnissen und Anwendungen gewahrt.

Umfang: Um aussagekräftige Ergebnisse zu erhalten, ist es unerlässlich, die gesamte oder zumindest einen Großteil der relevanten Literatur zu sichten. Um dabei eine gewisse Übersichtlichkeit zu gewährleisten, wurde die initial ermittelte Literatur systematisch gefiltert und gruppiert, da dies für die Ermittlung von Forschungslücken ausreichend ist.

Ausrichtung: Damit ein Überblick über bestehende Ansätze zur simulations- und szenariobasierten V&V gegeben werden kann, wurde eine methodische Einordnung der Ergebnisse vorgenommen. Historische Kontexte der gefundenen Arbeiten sind dabei nicht relevant. Da der übergreifende konzeptionelle Ansatz, wie er einleitend für diese Arbeit beleuchtet wurde, weitgehend anerkannt ist und konsistent genutzt wird, wird ausschließlich die Umsetzung des Konzepts betrachtet.

Zielgruppe: Die nachfolgende Vorstudie stellt eine Übersichtsarbeit dar und richtet sich daher in erster Linie an Forscher, die sich mit der V&V moderner maritimer Fahrsysteme beschäftigen.

10.1.2 SUCHSTRATEGIE

Aufgrund der Notwendigkeit, die vorhandene relevante Literatur annähernd in ihrer Gesamtheit zu ermitteln, stellt die Auswahl der Quellen für die Suche einen der kritischsten Aspekte der Planung eines Literaturreviews dar. Infolge der zunehmenden Digitalisierung von Forschungsergebnissen ist die

Tabelle 8: Für die Literatursuche verwendete Schlüsselwörter und die thematischen Merkmale aus denen diese abgeleitet wurden. Da die genutzten Datenbanken die Meta-Daten zu den indexierten Veröffentlichungen in englischer Sprache pflegen, wurden auch die Schlüsselwörter in englischer Sprache verfasst und genutzt.

Thematisches Merkmal	Abgeleitete Schlüsselwörter
Maritime Domäne	maritime, waterborne, vessel, ship
Simulations-basiert	simulation, simulative, simulation-based
Verifikation & Validierung	verification, validation, V&V, V+V, test
Verkehrs- & Fahrsysteme	traffic, traffic-system, transport, transport-system, autonomous, assistant, intelligent

Mehrzahl der Zeitschriften und Konferenzberichte inzwischen in Online-Datenbanken erfasst. Folglich wurden nur solche verwendet und auf eine Suche in physischen Bibliotheken, Sammelbänden etc. wurde verzichtet. Vom Brocke et al. [2009] raten dazu, Online-Datenbanken zu durchsuchen, die Zugang zu den führenden Zeitschriften des jeweiligen Fachgebiets bieten. Für die Durchführung der Suche wurden vier Forschungsdatenbanken verwendet: die multidisziplinären Datenbanken *Elsevier Scopus*[49] und *Web of Science (Core Collection)*[50] sowie die Informatik-fokussierte Volltext-Datenbank *IEEE Xplore*[51] [Cavacini, 2015]. Zusätzlich wurde aufgrund des umfangreichen Indexumfangs und der wachsenden Relevanz [Halevi et al., 2017] *Google Scholar*[52] genutzt. Auch die *ACM Digital Library*[53] wurde als potenzielle Datenquelle in Betracht gezogen, letztlich aber ausgeschlossen, da die Datenbank keinen für Benutzer zugänglichen Batch-Export der Suchergebnisse im CSV-Format oder ähnlichen Formaten unterstützt. Da die relevanten wissenschaftlichen Zeitschriften und Konferenzbände von den genannten Datenbanken bereits indexiert sind, wurden diese nicht weiter einzeln betrachtet. Auch Rückwärts- und Vorwärtssuchen unter Verwendung von Referenzen wurden nicht eingeplant.

In den genannten Datenbanken wurde eine Stichwortsuche durchgeführt. Um irrelevante Artikel von vornherein auszuschließen, wurden präzise Suchbegriffe und Schlüsselwörter verwendet. Die genutzten Schlüsselwörter wurden aus den Hauptmerkmalen des Themas abgeleitet. Verwandte Begriffe wurden jeweils zusätzlich inkludiert, um die Ergebnismenge zu erweitern. Die genutzten Schlüsselwörter sind Tabelle 8 zu entnehmen. Es wurden jeweils die vollständigen Metadaten, einschließlich des Titels, der Zusammenfassung und der zugewiesenen Schlüsselwörter, durchsucht, sofern die jeweilige Datenbank dies zuließ. Die einzelnen Schlüsselwörter innerhalb eines Merkmals wurden mit dem logischen Operator *OR* verknüpft und die Themengebiete untereinander mit dem logischen Operator *AND* kombiniert. Schlüsselwörter, die sich auf die Verwendung von Szenarien oder szenariobasiertem Testen beziehen, wurden explizit nicht berücksichtigt, da relevante Arbeiten existieren, die den Begriff *Szenario*, wie er in Abschnitt 7.2 definiert wurde, nicht verwenden oder nicht direkt erwähnen.

Um falsch positive Treffer zu minimieren, wurden Ausschlusswörter definiert. Diese wurden untereinander mit dem logischen Operator OR verknüpft und mit der Kombination aus den logischen Operatoren AND und NOT ausschließend an die zuvor beschriebenen Ausdrücke angehängt.

Ausschlusswörter: blood, numerical, pedestrians, car, racing, train, plane, ice, reactor, nuclear

Der Veröffentlichungszeitraum wurde nicht eingeschränkt, um keine möglicherweise relevanten Einträge auszuschließen. Aufgrund der Tatsache, dass entsprechende Technologien erst in den jüngeren

[49] `https://www.scopus.com/` [letzter Zugriff: 17.02.26]
[50] `https://www.webofscience.com/` [letzter Zugriff: 17.02.26]
[51] `https://ieeexplore.ieee.org/` [letzter Zugriff: 17.02.26]
[52] `https://scholar.google.de/` [letzter Zugriff: 17.02.26]
[53] `https://dl.acm.org/` [letzter Zugriff: 17.02.26]

vergangenen Jahren entstanden sind, ist eine gewisse Selbstbeschränkung aber vorhanden. Um die große Zahl irrelevanter Ergebnisse (insbesondere bei der Suche mit Google Scholar) zu begrenzen, wurden die Anfragen auf die 500 relevantesten Suchergebnisse pro Datenbank begrenzt.

10.1.3 SELEKTION

Sobald die potenziell relevanten veröffentlichten Arbeiten als direktes Ergebnis der Datenbanksuche vorliegen, ist es notwendig, diese auf ihre tatsächliche Relevanz zu prüfen [Kitchenham & Charters, 2007]. Im vorliegenden Fall wurde die gefundene Literatur daher in einem mehrstufigen Filterungsprozess mit zunehmender Vollständigkeit gesichtet und für den nächsten Schritt ein- oder ausgeschlossen. Zunächst wurden nur die Titel der Arbeiten gesichtet. Anschließend wurden die Zusammenfassungen und Fazits [Kitchenham & Charters, 2007] der verbleibenden Arbeiten gesichtet. Schließlich folgte eine Querlesung der verbleibenden Literatur und darauf aufbauend eine weitere Sortierung und Volltextlesung.

Einschlusskriterien

EK1 Veröffentlichung in einer wissenschaftlichen Zeitschrift oder einem wissenschaftlichen Konferenzband und Durchlaufen eines ordentlichen Peer-Review Verfahrens;

EK2 In englischer Sprache verfasst;

EK3 Bezug zu maritimem Verkehr und/oder maritimen Fahrzeuge;

EK4 Erwähnung (in den hinteren Filterstufen: Nutzung) von Simulationen;

EK5 Erwähnung von Validierung, Verifizierung oder Funktionstests automatisierter und autonomer Assistenz- und Fahrsysteme oder -funktionen;

Ausschlusskriterien

AK1 Maritimer Verkehr und/oder maritime Fahrzeuge im Titel, in der Zusammenfassung oder in den Schlüsselwörtern erwähnt, aber kein unmittelbarer Einsatz simulativer Testverfahren erkennbar;

AK2 Einsatz von makroskopischen Verkehrs-/Verkehrsflusssimulationen oder anderen zu abstrakten Beobachtungsebenen;

AK3 Exklusiver Fokus auf Unterwasserfahrzeuge;

AK4 Verwendung von Simulation, ohne Angaben dazu, welche genau und/oder zu den technischen Details dieser;

AK5 Duplikate;

AK6 Ergebnisse, die den in Tabelle 8 definierten Schlüsselwörtern entsprechen, aber trotzdem Falsch-Positive Ergebnisse darstellen (z. B. Übersichtsarbeiten, Studien über die Identifizierung relevanter Szenarien oder Studien über reine Umweltphänomene wie Eisdrift);

10.1.4 ANALYSE

Um ein fokussiertes Ergebnis zu gewährleisten, das dazu beiträgt, offene Herausforderungen zu identifizieren, wurde die Überprüfung konzeptzentriert durchgeführt [Webster & Watson, 2002]. Im Gegensatz zum autorenzentrierten Ansatz werden hierbei die identifizierten Arbeiten nach den verwendeten Konzepten gruppiert statt nach dem Autor. Dieser Ansatz wurde gewählt, da es dem Ziel zuträglich ist, die in der Forschung eingesetzten Anwendungen und Werkzeuge zu identifizieren und diese auf die Erfüllung der zuvor formulierten Anforderungen für den Einsatz in entwicklungsbegleitender szenariobasierter Verifizierung und Validierung zu überprüfen. Dieser Ansatz hebt somit die Schlüsselkonzepte und -technologien der im Einsatz befindlichen Simulationen hervor, fasst das vorhandene Wis-

sen zusammen und identifiziert bestehende Herausforderungen. Dementsprechend wurden die ausgewählten Veröffentlichungen in der Analysephase nach der Art der verwendeten Simulationsanwendung gruppiert und auf Grundlage der zuvor formulierten Anforderungen vergleichend bewertet.

10.2 DURCHFÜHRUNG

Die Datenbankabfragen wurden mit den in Abschnitt 10.1.2 definierten Schlüssel- und Ausschlusswörtern durchgeführt. Da nicht alle öffentlichen Schnittstellen der genannten Datenbanken die gleichen Möglichkeiten bieten, sind in Tabelle 9 einige Besonderheiten und Einschränkungen aufgeführt, die bei der Abfrage der jeweiligen Datenbank notwendig waren. Zusätzlich ist in derselben Tabelle die Anzahl der Ergebnisse pro abgefragter Datenbank angegeben. Das in *IEEE Xplore* verwendete Suchfeld *All Metadata* umfasst den Titel, die Zusammenfassung und die indizierten Schlüsselwörter und entspricht damit annähernd der Suche in *Scopus* und *Web of Science*. Lediglich *Google Scholar* lässt keine Suche über diese Felder zu, sodass stattdessen eine Suche über den Titel und den Volltext gewählt werden musste, was zu einer wesentlich größeren Ergebnismenge führte.

Die Suchergebnisse wurden, teilweise unter Zuhilfenahme externer Tools, exportiert und zusammengeführt. Die resultierende Ergebnismenge enthielt insgesamt 1452 Arbeiten. Duplikaterkennung und -entfernung wurden anhand von Titel und Autorennamen durchgeführt. Die verbleibenden 1221 einzigartigen Arbeiten wurden anschließend schrittweise, wie in Abschnitt 10.1.3 beschrieben, auf ihre Anwendbarkeit auf das vorliegende Thema überprüft. Die Anzahl der pro Schritt eingeschlossenen und ausgeschlossenen Arbeiten ist in Tabelle 10 dargestellt. Die hohe Anzahl an ausgeschlossenen Arbeiten ist darin begründet, dass in den identifizierten Veröffentlichungen zwar zahlreiche Simulationsanwendungen verwendet wurden, diese aber nicht immer für die weitere Untersuchung geeignet waren. So sind beispielsweise einige Simulationsanwendungen ausschließlich für die Darstellung des Verkehrs auf Flüssen in der Nähe von Brücken konzipiert. Zusätzlich fokussieren sich viele der gefundenen Arbeiten auf makroskopische Ergebnisse. Weil makroskopische Simulationen für das Testen automatisierter Fahrsysteme nicht geeignet sind, wurden diese Arbeiten aus der Ergebnismenge entfernt. Abgesehen von den Fokussierungen auf andersartige Fragestellungen und der damit einhergehenden

Tabelle 9: Anzahl der gefundenen Datensätze pro durchsuchter Datenbank.

Datenbank	Anmerkungen	#
Scopus	Titel, Zusammenfassung und Schlagworte durchsucht;	495
Web of Science	Titel, Zusammenfassung und Schlagworte durchsucht;	295
IEEE Xplore	Alle verfügbaren Metadaten durchsucht;	162
Google Scholar	Titel und Haupttext durchsucht; Die beiden letztgenannten ausschließenden Ausdrücke wurden entfernt (Limitierung auf 256 Zeichen); Die Ergebnisanzahl wurde auf 500 limitiert;	500

Tabelle 10: Statistik des mehrstufigen Selektionsprozesses.

Phase	Schritt	# beibehalten	# ausgeschlossen
Identifizierung	Suche	1452	n. a.
	Entfernung von Duplikaten	1221	231
Überprüfung	Überprüfung des Titels	125	1096
	Überprüfung der Zusammenfassung	33	92
Eignung	Querlesen	16	17
	Lesen des vollständigen Textes	15	1

Tabelle 11: Konzept-Matrix nach [Webster und Watson, 2002].

Name	Aktuellste Veröffentlichung	Art der Simulationsanwendung		
		Verteilt	Brücke	Eigenständig
Virtual Waterway	[Beschnidt und Gilles, 2005]			**X**
NSACA [54]	[Yang et al., 2007]	(X)	**X**	
SEAVIT [55]	[Tremori, Agresta und Ferrando, 2016]	**X**		
ASVTrafficSim	[Sauze, 2018]	(X)		**X**
HAGGIS	[Rüssmeier, Lamm und Hahn, 2019]	(X)		**X**
n. a.	[Yunsheng, Hui und Guofeng, 2020]	(X)		**X**
OSP [56,57,58]	[Smogeli et al., 2020]	**X**		
TestIT	[Pedersen et al., 2020]			**X**

X = zutreffend; (X) = teilweise zutreffend; leer = nichtzutreffend;

Nutzung eines ungeeigneten Betrachtungslevels oder -umfangs wurden auch Veröffentlichungen aufgrund mangelnder Klarheit in Bezug auf die verwendete Simulationsanwendung ausgeschlossen.

Dies zeigte sich besonders häufig in vagen Aussagen wie „Die Simulationsergebnisse werden gezeigt...“, „Die implementierte Simulation...“ oder „Die Simulation läuft...“, ohne dabei den tatsächlichen inhaltlichen und technischen Kontext zu spezifizieren. Veröffentlichungen mit einem zu spezifischen Fokus wurden ebenfalls ausgeschlossen. Letztlich wurden durch den zuvor beschriebenen Prozess 15 relevante Publikationen identifiziert. Diese wurden für den weiteren Verlauf der Untersuchung nach der verwendeten Simulationsanwendung gruppiert. Wurden mehrere Publikationen identifiziert, die eine bestimmte Simulationsanwendung beschreiben oder verwenden, so wird im weiteren Verlauf stellvertretend nur noch die zuletzt veröffentlichte genannt.

10.3 ERGEBNISSE

Wie in Abschnitt 10.1 beschrieben, wurden nicht die veröffentlichten Forschungsergebnisse selbst und die darin beschriebenen konkreten Anwendungsfälle betrachtet, sondern die zu Testzwecken verwendete und/oder entwickelte Simulationsanwendung. Dabei wurden die Veröffentlichungen zunächst nach dieser Simulation gruppiert, d. h., zwei oder mehr Veröffentlichungen, die den Einsatz derselben Simulation beschreiben, wurden zu einer Gruppe zusammengefasst. Anschließend wurden die verwendeten Simulationen anhand ihrer technischen Merkmale in drei übergreifende Gruppen eingeteilt: (1) klassische monolithische eigenständige Anwendungen, (2) verteilte Simulationsanwendungen und (3) physikalische Brückensimulatoren. Letztere werden klassischerweise vornehmlich zu Schulungszwecken. Tabelle 11 zeigt jeweils die jüngste Veröffentlichung für jede identifizierte Simulationsanwendung, das Jahr der Veröffentlichung dieser und die Einordnung in eines der drei genannten Konzepte. Um einen Überblick über die identifizierten Simulationen im Hinblick auf die generelle Eignung für den Einsatz im Rahmen der V&V von maritimen Fahrsystemen zu erhalten, wurden die in Abschnitt 9 definierten Anforderungen mit diesen abgeglichen. Dazu wurden die gefundenen Veröffentlichungen und, soweit vorhanden, öffentlich zugängliche Projektwebseiten und Ressourcen bzgl. der Erfüllung der Anforderungen ausgewertet. Das abschließende Ergebnis der Auswertung stellt Tabelle 12 dar: Die

[54] Simulation and Testing Platform for Navigation Safety and Automatic Collision Avoidance (NSACA)
[55] Sea Environment for Autonomous Vehicle Interoperability Testing (SEAVIT)
[56] Open Simulation Platform (OSP)
[57] https://opensimulationplatform.com/ [letzter Zugriff: 17.02.26]
[58] https://github.com/open-simulation-platform/ [letzter Zugriff: 17.02.26]

Tabelle 12: Erfüllung der in Abschnitt 9 formulierten Anforderungen durch die identifizierten, zu Testzwecken eingesetzten maritimen Simulationsanwendungen.

Konzept	Veröffentlichung	Anforderungen												
		Simulation (A1)						Szenario (A2)						
		1	2	3	4	5	6	1	2	3	4	5	6	7
Monolithisch	[Pedersen et al., 2020]	?	(X)	?	**X**	(X)	?	**X**	?		(X)	?	?	?
	[Yunsheng, Hui und Guofeng, 2020]	**X**	(X)		(X)	(X)	**X**	**X**	?		(X)	?	X	?
	[Sauze, 2018]		(X)		(X)			(X)	?			?	?	
	[Beschnidt und Gilles, 2005]	**X**	(X)		**X**	**X**	**X**	**X**	(X)	?	(X)	**X**	**X**	?
Brücke	[Yang et al., 2007]		(X)		**X**	**X**	?	**X**	?		?	?	(X)	?
Verteilt	[Smogeli et al., 2020]	(X)	**X**	**X**	**X**	(X)	(X)	**X**	(X)	(X)	(X)	**X**		
	[Rüssmeier, Lamm und Hahn, 2019]	**X**	(X)		**X**	**X**	X	**X**	(X)	(X)	(X)	**X**	(X)	**X**
	[Tremori, Agresta und Ferrando, 2016]	(X)	**X**	**X**	(X)	?	**X**	**X**	?		(X)	?		?

X = vollständig erfüllt; (X) = teilweise erfüllt; ? = nicht spezifiziert; leer = nicht erfüllt;

Veröffentlichung-Simulation-Paare sind auf der linken Seite anhand der verwendeten Konzepte gruppiert aufgeführt und die Anforderungserfüllung ist auf der rechten Seite dargestellt.

Es ist offensichtlich, dass keine der aktuell in entsprechenden wissenschaftlichen Veröffentlichungen zu Testzwecken eingesetzten maritimen Simulationen, alle aufgestellten Anforderungen erfüllt. Überdies kann für zahlreiche Anforderungen der Erfüllungsgrad nicht bestimmt werden, da es an öffentlich zugänglichen Ressourcen mangelt und die veröffentlichten schriftlichen Ergebnisse häufig nicht genügend technische Details enthalten. Der Mangel an Details der technischen Umsetzung, zeigt sich insbesondere in Bezug auf die Art und Weise, wie Szenarien beschrieben und modelliert werden, um als Ausgangspunkt für den simulativen Test zu dienen. Dies deutet darauf hin, dass diesem speziellen Bereich nicht genügend Aufmerksamkeit geschenkt wurde. Auch die Modellierung, Speicherung, Übertragung und Nutzung von Szenarien ist von entscheidender Bedeutung für die effektive Umsetzung von szenariobasierten V&V-Methoden (vgl. Abschnitte 2 und 7). Es scheint jedoch, als wäre dies kein exklusives Problem der maritimen Domäne, sondern vielmehr wird in Bezug auf den Einsatz von Simulationen zu Testzwecken in allen Verkehrsdomänen der Spezifikation von Testszenarien regelmäßig wenig bis keine Beachtung geschenkt. So schneidet etwa ein aktueller und ansonsten sehr umfangreicher Überblick über Simulationswerkzeuge für das Testen autonomer Straßenfahrzeuge [Zhang et al., 2024] den Themenkomplex der Szenarien lediglich in Bezug auf einen Vergleich der realitätsnahen Darstellungsmöglichkeiten der jeweiligen Simulationsanwendung an. Abschließend wird aber darauf hingewiesen, dass ein Fokus auf Tests aussagekräftiger Szenarien gelegt werden muss – dem Kerngedanken des szenariobasierten Testens.

Ein weiterer wenig bis gar nicht beachteter Bereich ist die nahtlose Integration der zu testenden Systeme und Komponenten selbst. Häufig werden Simulationsanwendungen speziell für die Bewertung autonomer Navigations- und Fahrmodelle entwickelt, was einen Spezialfall von Software-in-the-Loop-Tests darstellt. Typischerweise stellen die Simulationsseitigen Schnittstellen dabei navigationsrelevante Sensordaten zur Verfügung und nehmen ausschließlich direkte Steuerbefehle entgegen. Für den Test von Systemen, die keinen direkten Einfluss auf die Steuerung haben, wie Teilsysteme mit untergeordneter Rolle in der Navigation oder Warnsysteme, sind sie nicht geeignet.

Erwähnenswert ist, dass verteilte Simulationen den höchsten Erfüllungsgrad der Anforderungen an Simulationsanwendungen (A1) erreichen. Daraus lässt sich schließen, dass diese in der zukünftigen Forschung und Entwicklung vorrangig behandelt werden sollten. Ihr dezentraler Charakter ermöglicht konzeptbedingt die relativ unkomplizierte Integration externer Systeme in die Simulation. Ein externes System oder eine externe Komponente kann somit entweder als Teilnehmer oder als zu testendes System mit geringem Anpassungsaufwand am Simulationslauf teilnehmen.

Eine wesentliche Herausforderung für den weitverbreiteten Einsatz von szenariobasierten Simulationsläufen zur Verifikation und Validierung (V&V) von maritimen Fahrsystemen, die nicht unmittelbar aus der Tabelle ersichtlich ist, aber bei der Analyse deutlich wurde, ist die enge Interdependenz zwischen dem Simulationsmodell und dem Szenariomodell: Verfügt eine Simulation über ein eigenes internes Datenmodell und ist in der Lage, Szenarien einlesen zu können, welche eine parametrisierte Teilmenge dieses Datenmodells beschreiben, so ist die Korrektheit des Szenarios von dem Datenmodell der Simulation abhängig. Ein Szenario muss die Fähigkeiten des Datenmodells der Simulation widerspiegeln und umgekehrt, um Informationsverluste oder Fehlinterpretationen zu vermeiden. Diese Abhängigkeit muss bei zukünftigen Entwicklungen minimierend berücksichtigt werden.

Barceló hat bereits 2010 darauf hingewiesen, dass trotz der global hohen Relevanz und der nicht geringfügigen Menge veröffentlichter Arbeiten mit Bezug zu Verkehrssimulationen unterschiedlicher Ausprägungen eine einheitliche Betrachtung der Anwendung solcher fehlt:

> „*[…] traffic simulation still lacks a unified treatment. Dozens of papers on theory and applications are published […] every year. […] Yet, the only comprehensive treatment of the subject to be found so far is in the user manuals of various software products.*“[59]
>
> [Barceló, 2010b]

Spätestens bei einem vergleichenden Blick auf den Automobilbereich wird deutlich, dass es im maritimen Sektor noch immer einen bemerkenswerten Mangel in Bezug auf die Standardisierung der Beschreibung und Modellierung von Szenarien und der dafür nötigen ganzheitlichen Betrachtung des Einsatzes von maritimen Verkehrssimulationen gibt. Im Bereich kleinteiliger Simulation von Straßenfahrzeugen haben sich u. a. seit der Veröffentlichung der oben zitierten Aussage die Formate OpenSCENARIO, OpenDRIVE und OpenCRG [Fan & Krishnan, 2020] etabliert und werden von zahlreichen Simulationsanwendungen unterstützt. Diese Standards bieten eine einheitliche Basis, die eine schnelle Entwicklung, die Vergleichbarkeit der Ergebnisse und eine erfolgreiche Zusammenarbeit von Projektpartnern ermöglicht. Ohne ähnliche Standards für maritime Simulationen werden sich szenario- und simulationsbasierte Ansätze wahrscheinlich nicht durchsetzen können.

11 HANDLUNGSBEDARF

In Teil I dieser Arbeit wurde einleitend die Bedeutung einer Szenario- und Simulations-basierten Verifizierung & Validierung (V&V) für die öffentliche Akzeptanz und flächendeckende Einführung von softwarebasierten automatisierten maritimen Fahrsystemen herausgearbeitet. Anschließend wurden zu erfüllende Anforderungen an entsprechende Simulationsanwendungen anhand der Aufbereitung grundlegender Konzepte und Vorgehensmodelle der Kernbereiche einer solchen und eines Blicks auf Ergebnisse verwandter Verkehrsdomänen gezielt abgeleitet. In Abschnitt 10 wurde eine sorgfältig ge-

[59] Auslassungen;

plante und ausführlich dokumentierte systematische Literaturrecherche unter Zuhilfenahme einiger der größten relevanten wissenschaftlichen Online-Literaturdatenbanken durchgeführt, um existierende Simulationsanwendungen zu identifizieren, die aktiv in Forschungsarbeiten eingesetzt werden. Die gefundenen Anwendungen wurden anschließend eingehend analysiert und den formulierten Anforderungen aus Abschnitt 9 gegenübergestellt. Als Ergebnis dieser umfassenden, systematischen und integrativen Untersuchung wurden Forschungslücken identifiziert, die geschlossen werden müssen, um einen erfolgreichen Betrieb von automatisierten und autonomen Schiffen in der realen Welt durch den Einsatz von entwicklungsintegrierter, simulations- und szenariobasierter V&V zu erreichen.

Die folgenden inhaltlichen Empfehlungen für künftige Forschungsaktivitäten werden unter der Annahme vorgeschlagen, dass sie zu gezielteren Ergebnissen führen werden. Die Empfehlungen haben einen allgemeingültigen Charakter für zukünftige Forschungsaktivitäten und haben als Ergebnis der empirischen Phase dieser Arbeit (vgl. Abschnitt 5) eine eigenständige Daseinsberechtigung. Angesichts dessen sind sie hier trotz der nachfolgenden direkten Überführung in konkrete Ziele für die praktische Phase dieser Arbeit als relevantes Teil-Ergebnis dieser Arbeit aufgelistet:

E1 Künftige Forschungs- und Entwicklungsanstrengungen müssen sich darauf fokussieren, den gesamten V&V-integrierten Entwicklungsprozess in jeder Phase zu unterstützen.

E2 Für die breite Einführung von simulations- und szenariobasierten V&V-Methoden ist ein einheitliches Format für die technische Modellierung von Seeverkehrsszenarien unerlässlich. Ein solches würde auch die Zusammenarbeit und Kommunikation zwischen Projekt- und Entwicklungspartnern fördern.

E3 Es muss genauer untersucht werden, wie ein Szenario technisch so spezifiziert werden kann, dass eine Simulationsanwendung ein gegebenes Szenario mit unterschiedlichen Komplexitäts- und Abstraktionsgraden ohne Informationsverlust ausführen kann.

E4 Neben umfassender technischer Funktionalität muss auch die effiziente Einsetzbarkeit als entwicklungsbegleitendes Werkzeug im Fokus zukünftiger Entwicklungen stehen.

E5 Die Möglichkeit, das Testobjekt als Teil des zu simulierenden Szenarios so früh wie möglich zu integrieren, muss untersucht und realisiert werden, um zeitaufwendige individuelle Anpassungen an der Simulationsanwendung selbst für jedes einzelne System-unter-Test zu vermeiden.

E6 Die Interoperabilität zwischen Simulationsanwendungen und externen Komponenten und Systemen sollte stärker fokussiert werden.

III

Komponentenbasierte Modellierung und Simulation von Verkehrsszenarien

In der vorangegangenen explorativen Phase wurden die unscharfen und vielfältigen technischen sowie inhaltlichen Herausforderungen der Verifizierung und Validierung automatisierter Fahrfunktionen und Systemkomponenten moderner Schiffe klar abgegrenzt und benannt. Durch eine ausführliche Erkundung des Forschungsfelds inklusive grundlegender Begrifflichkeiten, Technologien und Prozesse konnten Anforderungen identifiziert werden, die es durch eine entsprechende Modellierungs- und Simulationsarchitektur zu erfüllen gilt. Auch konnte gezeigt werden, dass die konkreten Computersimulationswerkzeuge, die im Rahmen zuletzt veröffentlichter wissenschaftlicher Arbeiten eingesetzt wurden, diesen Anforderungen nicht entsprechen. Ebenso konnte festgestellt werden, dass es diesbezüglich kein einheitliches Vorgehen gibt und die Hürde für den systematischen Einsatz von Simulationsanwendungen als Testwerkzeug hoch zu sein scheint. Somit haben sich dringende Handlungsbedarfe ergeben, die es in die Tat umzusetzen gilt.

Dieser Teil der Arbeit widmet sich der Gestaltung einer softwaretechnischen Lösung, welche die formulierten Handlungsempfehlungen adressiert, um die identifizierten Anforderungen zu erfüllen. Zur Ermöglichung eines zielgerichteten Vorgehens werden dafür in Abschnitt 12 zunächst auf Basis der Erkenntnisse des vorigen Teils dieser Arbeit konkrete gestaltungsorientierte Ziele formuliert sowie eine Hypothese formuliert, die es durch die konstruktive Erreichung der Ziele zu überprüfen gilt.

Im nachfolgenden Abschnitt 13 wird zunächst untersucht, welche Aufgaben überhaupt im Rahmen eines Entwicklungsprozesses mit entwicklungsbegleitend integrierten szenariobasierten simulativen Tests anfallen und welchen Rollen diese zugeordnet werden können. Die so geschaffene klare Trennung von Verantwortlichkeiten wird anschließend genutzt, um zielgerichtete, lösungsbezogene Anforderungen mit Bezug zu den zuvor formulierten Zielen zu erarbeiten.

Abschnitt 14 beginnt anschließend mit der Vorstellung der zur Erfüllung der formulierten Anforderungen entworfenen Lösungskonzepte. Die Betrachtungsebene wandert dabei vom Groben zum Feinen, d. h., es werden zunächst die übergreifenden Lösungsideen vorgestellt und anschließend inhaltlich konkret ausgestaltet. Zwischengeschoben wird zusätzlich eine abgrenzende Vorstellung von Arbeiten, die in direktem Bezug zu Teilen des vorgestellten übergreifenden Lösungskonzeptes stehen.

Als Grundlage für die in Abschnitt 16 durchgeführte Überprüfung der Ergebnisse auf Anforderungserfüllung, Eignung für den Einsatz im betrachteten Problembereich sowie Erreichung der formulierten Ziele wird in Abschnitt 15 die Überführung der Lösungskonzepte in ein softwaretechnisches Artefakt beschrieben. Während dieses Prozesses getroffene Entscheidungen werden im Rahmen dessen begründet dargestellt und der entstandene Prototyp als Resultat der Überführung vorgestellt.

12 GESTALTUNGSORIENTIERTE ZIELSETZUNG

Um der ingenieurwissenschaftlichen Aufgabe, ein neuartiges technisches Artefakt zur Lösung eines konkreten Problems zu konstruieren, gerecht zu werden, bedarf es zunächst der Formulierung einer klaren und überprüfbaren Zielsetzung. Nachfolgend werden daher konkrete, gestaltungsorientierte Ziele für die praktische Phase der vorliegenden Arbeit aus den Ergebnissen der vorigen Teile abgeleitet. Dabei adressiert jedes Ziel eine der in Abschnitt 11 formulierten Handlungsempfehlungen (vgl. Abbildung 40) und somit eine der im Rahmen der explorativen Phase identifizierten Lücken.

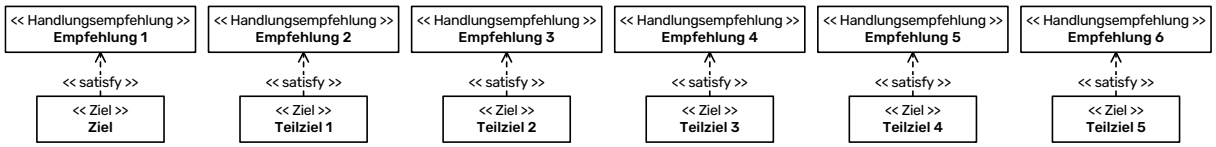


Abbildung 40: Anforderungsdiagramm zur Darstellung der direkten Beziehung jedes gestaltungsorientierten Ziels zu einer der formulierten Handlungsempfehlungen.

Rückblickend auf Teil II dieser Arbeit bleibt festzustellen, dass keine der identifizierten Simulationsanwendungen alle Eigenschaften in vollem Umfang besitzt, die es für eine entwicklungsbegleitende V&V maritimer softwarebasierter Fahrsysteme und -komponenten benötigt. Zudem scheint die Akzeptanz des Einsatzes von szenariobasierten simulativen Testläufen in der Forschung zur Automation maritimer Fahrzeuge gering zu sein. Als weitere Forschungsagenda ist daher die Neuentwicklung und Erprobung eines softwaretechnischen Konzepts für die Spezifikation und Simulation von Szenarioinstanzen erforderlich. Das übergeordnete, gestaltungsorientierte Ziel des konstruktiven Teils dieser Arbeit lässt sich daher wie folgt formulieren:

Ziel (Z0) Entwicklung eines softwaretechnischen Konzepts, welches eine einheitliche szenariobasierte Durchführung submikroskopischer Simulationsläufe zur Verifizierung und Validierung softwarebasierter maritimer Fahrsysteme entwicklungsbegleitend ermöglicht.

Dieses übergeordnete Ziel adressiert die identifizierte Problemstellung in Bezug auf die Leitfrage. Sie trifft jedoch noch keine Aussagen darüber, welche Eigenschaften das nötige Softwareartefakt haben muss, um das Ziel zu erreichen. Durch eine Abbildung des in Teil II dieser Arbeit gesammelten Wissens auf die durch das übergeordnete Ziel (Z0) adressierten thematischen Teilbereiche sowie die Teilfragen 1–3 lassen sich spezifischere Teilziele (Z1–Z5) ableiten, welche nachfolgend aufgelistet sind. Durch die feinere Granularität lassen sich die Teilziele auf Erreichung prüfen und bilden gemeinsam das übergeordnete Ziel (Z0) vollständig ab. Die Teilziele beschreiben zudem in erster grober Annäherung die Eigenschaften, die ein Modellierungs- und Simulationswerkzeug innehaben muss, damit dieses ganzheitlich entwicklungsprozessbegleitend zur Unterstützung der V&V automatisierter softwarebasierter Fahrzeugkomponenten eingesetzt werden kann.

Teilziel 1 (Z1) Entwicklung einer einheitlichen Basis zur Erstellung flexibel wiederverwendbarer und erweiterbarer Modelle maritimer Verkehrsszenarien sowie zur Spezifikation von Instanzen dieser Modelle.

Das Teilziel 1 ergibt sich aus der Teilfrage 1. Aufgrund der Abhängigkeit des Modellzwecks vom konkreten Anwendungs-/Testfall und der Tatsache, dass der Modellzweck unmittelbar die Modellformu-

lierung und den Modellumfang bestimmt [Barceló, 2010a; Bossel, 2004], ist weder eine Eins-zu-eins-Abbildung von System zu Modell noch ein singuläres allumfassendes Modell sinnvoll (vgl. Abschnitt 7.3.1). Gleichzeitig ist eine der zentralen Herausforderungen szenariobasierter simulativer Überprüfungen von Fahrsystemen die große Variabilität der Anforderungen an die Bestandteile eines Szenarios [Klikovits et al., 2024]. Letzteres wird durch den angestrebten Einsatz während des gesamten Entwicklungsprozesses zusätzlich verstärkt. Eine Eigenschaft des Modells muss also *Flexibilität* bzgl. der zu testenden Komponente, des Testfalls und der konkret zu beantwortenden Frage sein. Gleichzeitig muss die Erstellung von Szenariomodellen effizienter werden, um den kontinuierlichen Einsatz während der Entwicklung zu ermöglichen. Dies kann durch die *Wiederverwendbarkeit* von (Teil-)Modellen und die Möglichkeit der *Erweiterung* bestehender (Teil-)Modelle geschaffen werden. Damit (Teil-)Modelle miteinander kompatibel sind, bedarf es einer *einheitlichen Basis* (vgl. Abschnitt 7.4.3), auf welcher die Modelle aufbauen. Dass *Szenarioinstanzen* im Sinne der Wiederverwendung zudem getrennt von der Simulationsanwendung *persistier- und austauschbar* sein sollten, zeigt die weite Akzeptanz und Verbreitung der erfolgreichen Spezifikationsstandards im Automotive Bereich.

Teilziel 2 (Z2) Entwicklung einer Anwendung zur direkten Durchführung simulativer Testläufe, basierend auf Instanzen heterogener Szenariomodelle.

Teilziel 2 ergibt sich ebenfalls aus der Teilfrage 1, bezieht sich aber auf die Simulationsanwendung als den modellausführenden Teil. Die Notwendigkeit der *Flexibilität* aufgrund der direkten Beziehung zwischen dem konkreten Testfall und dem Modellzweck ergibt sich hier ebenfalls, da die Inhalte des Szenariomodells Anforderungen an das Simulationsmodell stellen [Klikovits et al., 2024] (vgl. Abschnitt 7.4.2). Bezüglich der Beziehung zwischen Szenario- und Simulationsmodell muss daher Kohärenz geschaffen werden: Änderungen am Szenariomodell müssen ohne manuellen Arbeitsaufwand an das Simulationsmodell propagiert werden [Dahanayake & Thalheim, 2010], damit effiziente Flexibilität möglich ist. Es muss daher möglich sein, alle Szenarioinstanzen, die aufbauend auf der in Teilziel 1 erwähnten Basis erstellt wurden, *direkt in einen Simulationslauf zu überführen*. Daraus lässt sich schlussfolgern, dass die in Abschnitt 7.4.3 beschriebene orthogonale Beziehung der beiden beteiligten Modellierungsräume aufgelöst werden muss.

Teilziel 3 (Z3) Entwicklung eines Konzepts zur Kapselung der Komplexitäten zugrunde liegender abstrakter Strukturen der Modelle und technischer Abläufe der Simulationsanwendung.

Dieses Ziel ergibt sich direkt aus der Teilforschungsfrage 3. Die Spezifikation von Szenarien sowie die simulative Ausführung dieser (vgl. Teilziel 1 und Teilziel 2) darf nicht das Vorhandensein von Wissen über zugrundeliegende Details, wie umfassende Wirkzusammenhänge der Verkehrsdomäne (vgl. [Barceló, 2010a]) oder technische Abläufe der Simulationsanwendung (vgl. [Ramos & Piera, 1999]), voraussetzen. So soll letztlich nicht nur die Komplexität selbst, sondern auch der Zeitaufwand von simulativen, szenariobasierten Testläufen reduziert werden. Vor allem für den entwicklungsbegleitenden Einsatz (vgl. Abschnitt 7.1.2) darf der für die Spezifikation von Szenarien und Simulation dieser nötige Aufwand nicht zu hoch sein, damit die eigentliche Entwicklung nicht verzögert wird.

Teilziel 4 (Z4) Entwicklung eines Konzepts zur Integration heterogener zu testender Komponenten in Instanzen des Szenariomodells.

Nachdem in Teil II dieser Arbeit festgestellt wurde, dass Szenario- und Simulationsmodelle zwar in einer wechselseitigen Beziehung zueinander stehen, aber trotzdem voneinander getrennt zu betrachten sind, ergeben sich aus Teilfrage 3 analog zu den Teilzielen 1 und 2 zwei Ziele. Teilziel 4 hat dabei, als erstgenanntes von diesen beiden, die Integration in das Szenario zum Inhalt. Im vorliegenden Fall ist die Simulation von Verkehrsszenarien kein reiner Selbstzweck. Stattdessen steht das Testen einer externen Komponente im Fokus. Um dieser Ausrichtung gerecht zu werden, muss eine Möglichkeit geschaffen werden, dass zu testende Komponenten in unterschiedlichen möglichen Entwicklungszuständen (vgl. Model-/Software-/Hardware-in-the-Loop, Abschnitt 7.3.4) analog zu den Modellinhalten als Teil der Szenariospezifikation abgebildet werden.

Teilziel 5 (Z5) Entwicklung eines Konzepts zur aktiven Einbindung heterogener externer Komponenten in den durch eine Szenarioinstanz repräsentierten Simulationslauf.

Das an Teilziel 4 anknüpfende Teilziel 5 weist bezüglich seiner Ausrichtung Überschneidungen mit Teilziel 3 auf: die Vermeidung der Konfrontation des Benutzers mit Komplexitäten, die über die Spezifikation von Szenarien und die simulative Durchführung dieser hinausgehen. Für den durchgehenden Einsatz über den gesamten Entwicklungsprozess hinweg sowie die Austauschbarkeit von Szenarien und Modellen muss möglichst vermieden werden, dass für die Kommunikation mit externen Komponenten Anpassungen an der Simulationsanwendung selbst nötig sind (vgl. [Kühnel et al., 2009]). Die Simulationsanwendung muss daher anhand der Informationen aus der Szenarioinstanz eine Kommunikation mit der abgebildeten externen Komponente initiieren und mit dieser während des Simulationslaufs im Sinne einer *Closed Loop Simulation* bidirektional Daten austauschen können.

Durch die Ableitung der fünf Teilziele aus jeweils einer der drei Teilfragen ergeben sich entsprechend drei thematische Bereiche (vgl. Abbildung 41). Trotz gewisser Eigenständigkeit dieser Bereiche ist eine vollständig getrennte Bearbeitung nicht sinnvoll, da die Konzepte und ihre Ausgestaltung gegenseitigen Einfluss aufeinander haben. Die Arbeit an den Konzepten und der prototypischen Umsetzung selbst erfolgte daher iterativ. Die Vorstellung der Ergebnisse in den Abschnitten 14.1 und 14.3 erfolgt aber aus Gründen der Verständlichkeit und Lesbarkeit weitestgehend linear. Stattdessen wird an nötiger Stelle mit Blicken in die anderen konzeptuellen Themenbereiche durch Querverweise gearbeitet.

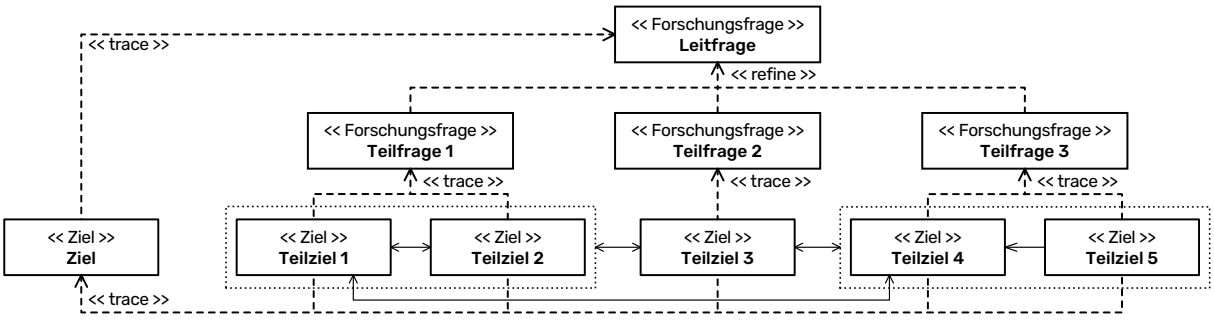


Abbildung 41: Anforderungsdiagramm zur Darstellung der hierarchischen Abhängigkeiten zwischen den zu Beginn aufgestellten Forschungsfragen und den Zielen, die zur Beantwortung dieser im praktischen Teil verfolgt werden.

Auf Grundlage der in Teil II dieser Arbeit gewonnenen Erkenntnisse und der formulierten gestaltungsorientierten Ziele als erste grobe Lösungsansätze wird die folgende Hypothese formuliert, deren Beweis das Erreichen der formulierten Teilziele bedingt und die erarbeiteten Konzepte als Antwort auf die Forschungsfrage bestätigt:

Hypothese

Die Auflösung der orthogonalen Beziehung zwischen Szenario- und Simulationsmodellen in Kombination mit einer Kapselung der Simulations- und Modellierungskomplexität ermöglicht ein Softwarewerkzeug, das für die simulative Unterstützung entwicklungsbegleitender Verifizierung und Validierung ausreichend flexibel und effizient einsetzbar ist.

13 LÖSUNGSBEZOGENE ANFORDERUNGEN

In der höchst dynamischen Landschaft aktueller wissenschaftlicher Forschung ist die Entwicklung von robuster und zuverlässiger Software unverzichtbar. Diese Softwarewerkzeuge dienen als integrale Bestandteile im Arsenal des Forschers und erleichtern die komplexe Analyse, Simulationen, Evaluation und Interpretation von experimentellen Ansätzen und Ideen. Das übergeordnete Ziel solcher Software ist nicht nur die Rationalisierung wissenschaftlicher Arbeitsabläufe, sondern auch, insbesondere für Simulationsanwendungen zur Überprüfung neuer Konzepte und Umsetzungen, ein Beitrag zur Überprüfbarkeit und Transparenz von Ergebnissen. In diesem Zusammenhang ist die Anforderungsspezifikation eine obligatorische Prozessphase, auf der die gesamte anschließende ingenieurswissenschaftliche Lösungsfindung aufbaut.

Sowohl System- als auch Software-Engineering im Allgemeinen umfasst einen systematischen, disziplinierten und quantifizierten Ansatz für die Entwicklung, die Wartung und den Betrieb von Softwareanwendungen, Komponenten und Systemen. Ein wesentlicher Aspekt dieses Prozesses ist die Anforderungsanalyse, die das Ermitteln, Dokumentieren, Testen und Verwalten der Anforderungen für das zu entwickelnde Produkt umfasst. [Herrmann, 2022; ISO/IEC/IEEE, 29148:2018; Koelsch, 2016] Um dieser komplexen Zusammenhänge Herr zu werden, ist es unerlässlich, strukturiert vorzugehen und Erkenntnisse aus vorhandener Literatur, veröffentlichten wissenschaftlichen Ergebnissen, bestehenden Softwaretools, -anwendungen und -systemen sowie bewährten Verfahren zu gewinnen (vgl. Teil II). Aufgrund der Neuartigkeit und der damit einhergehenden technischen Unwägbarkeiten ist ein umfassendes *Upfront-Design* – also die Erstellung eines detaillierten Entwurfs auf der Grundlage vollständig dokumentierter Anforderungen – jedoch nicht realisierbar. Daher werden zunächst nur die gestaltungsorientierten Anforderungen dokumentiert, die in der frühen Projektphase erfasst werden können, und das System wird schrittweise im Rahmen der Konzeptphase weiterentwickelt.

Darüber hinaus können die Notwendigkeit zur Einhaltung gesetzlicher Normen, zur Interoperabilität mit bestehenden Tools sowie Überlegungen zur Benutzerfreundlichkeit und -erfahrung zu einer zusätzlichen Komplexität des Anforderungsspezifikationsprozesses führen. Die Software muss nicht nur der Zielerreichung der jeweiligen Nutzergruppe innerhalb der wissenschaftlichen Gemeinschaft entsprechen, sondern sich auch nahtlos in die Arbeitsabläufe der potenziellen Anwender einfügen und Zugänglichkeit gewährleisten. Insbesondere der letztgenannte Punkt wird häufig in der Entwicklung unterstützender Softwaretools und -systeme übersehen oder aktiv ausgeblendet, obwohl eben jene

Höhe der Einstiegshürde darüber entscheiden kann, ob die Software eine breite Anwendung findet oder trotz hohen Potenzials ungenutzt bleibt und keine Relevanz gewinnen kann (vgl. [Tolk, 2002]).

13.1 NUTZERGRUPPEN SZENARIOBASIERTER SIMULATIONSLÄUFE

Drogoul et al. schlugen bereits im Jahr 2003 als mögliche Lösung von wiederkehrenden Problemen bei der Überführung von theoretischen Modellen in ausführbare programmatische Modelle vor, dass zunächst betrachtet werden müsse, *was* von *wem* innerhalb eines Simulationslebenszyklus erstellt wird. Erst danach könne man die Fragen nach dem *Wann* und dem *Wie* angehen. Sie identifizieren und beschreiben in diesem Zusammenhang drei Rollen, die zusammenarbeiten müssen, um ein Simulationsmodell zu entwerfen, umzusetzen und es simulativ auszuführen: *der Thematiker* (engl.: the thematician), *der Informatiker* (engl.: the computer scientist) und *der Modellierer* (engl.: the modeler). In dieser Rollenverteilung besteht das Ziel des Informatikers hauptsächlich darin, die Simulationsanwendung in Form eines Computerprogramms zu erstellen, zu pflegen und ggf. zu erweitern. Er besitzt somit detailliertes Fachwissen über Simulationstechniken, konkrete Technologien und prozedurale Abläufe. Der Thematiker auf der anderen Seite besitzt Fachwissen über die zu simulierende Domäne und ihre Inhalte. Er bestimmt die Absicht des Simulationsprozesses, d. h. die inhaltliche Komponente der Abbildung von Dingen der echten Welt auf ein Simulationsmodell. Dabei wird deklaratives Domänenwissen eingebracht, um das generelle Verhalten der zu erstellenden Modelle zu definieren, werden beobachtungsbasierte Erfahrungen genutzt, um Parameter, Ausgangszustände und Einschränkungen festzulegen, sowie die anhand des Modells zu beantwortenden Fragen formuliert. Die Aufgabe des Modellierers ist es dann, die konzeptuelle Lücke zwischen Thematiker und Informatiker zu schließen und die Vorgaben des Thematikers in eine Form zu überführen, die durch den Informatiker verstanden, in die Simulationsanwendung integriert und ausgeführt werden kann. [Drogoul et al., 2003]

Auch im Handbuch zum *Simulation Model Portability* (SMP) Standard der *European Space Agency* (ESA) wird eine ähnliche Einteilung von Beteiligten in Rollen vorgenommen: *Simulator Designers*, *Model Developers*, *Tool Developers*, *Environment Furnishers* [ESA, SMP2]. Ramos & Piera schlussfolgern hingegen lediglich, dass während des Modellierungsprozesses und in den resultierenden Modellen deklaratives Wissen von prozeduralem Wissen vollständig getrennt sein muss [Ramos & Piera, 1999]. In [Bossel, 2004] wird eine Aufgabenverteilung bei der Erstellung wirkstrukturbeschreibender Modelle implizit über die nötige Zusammenarbeit von *Modellentwicklern* und *intimen Systemkennern* angesprochen. Auch Barceló [2010a] beschreibt einen mehrstufigen Modellierungsprozess: Durch eine Systemanalyse wird ein konzeptionelles Modell erstellt, welches durch einen Übersetzungsprozess in ein Computermodell überführt wird, welches anschließend kalibriert werden muss, bevor die simulativen Tests durchgeführt werden können.

Aus Teilziel 3 ergibt sich die Notwendigkeit der eindeutigen Benennung derjenigen Akteure des V&V-integrierten Entwicklungsprozesses unter Einsatz von szenariobasierten Simulationsläufen, die an der Spezifikation von Szenarioinstanzen beteiligt sind. Die Anforderungen an die zu erstellende Lösung können anschließend gezielt so formuliert werden, dass sie eine Kapselung der Komplexitäten entsprechend der Aufgabengebiete der betroffenen Akteure fordern, mit dem Ziel, die nötige Effektivität des Einsatzes sowie einen möglichst niedrigschwelligen Einsatz dieser zu erreichen (vgl. letzter Absatz der Einleitung zu Abschnitt 13). Auch nach gründlicher und umfangreicher Recherche konnte keine solche Analyse zu den Akteuren und ihren Rollen in der Durchführung szenariobasierter simulativer Testrei-

hen in der gängigen wissenschaftlichen Literatur gefunden werden. Deshalb wurde der in Abbildung 2 gezeigte Prozess weiter konkretisiert und angelehnt an [Drogoul et al., 2003] bzgl. der beteiligten Akteure analysiert. Es wurden insgesamt acht Aktivitäten und sechs Akteure identifiziert. Diese sind in Abbildung 42 in Form eines UML-Anwendungsfalldiagramms dargestellt. Von diesen sind vier Aktivitäten und drei Nutzergruppen von Relevanz für diese Arbeit und werden nachfolgend genauer beschrieben.

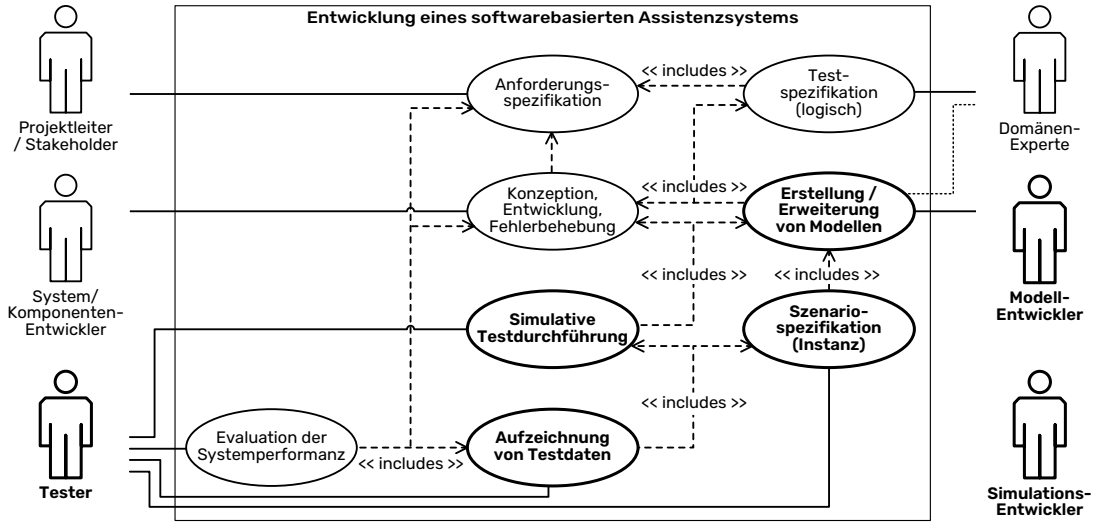


Abbildung 42: Anwendungsfalldiagramm zur Darstellung der Rollen und Tätigkeiten, die Teil eines V&V-integrierten Entwicklungsprozesses beim Einsatz von szenariobasierten Simulationsläufen sind, und der Beziehungen zwischen ihnen. Die dargestellten Tätigkeiten stellen eine Verfeinerung des zu Beginn dieser Arbeit in Abbildung 2 dargestellten allgemeinen szenariobasierten Entwicklungsprozess dar.

Simulationsentwickler: Der Simulationsentwickler hat mit dem eigentlichen entwicklungsbegleitenden Testprozess keine Berührungspunkte. Seine Aufgabe ist es, die Simulationsanwendung zu entwerfen, umzusetzen, zu pflegen und bei Bedarf anzupassen. Er muss theoretisches Wissen über Simulationstechniken besitzen sowie praktisches Wissen über die verwendeten konkreten Simulationstechnologien, -standards und Schnittstellen genutzter Frameworks und die prozeduralen Abläufe. In Hinblick auf Teilziel 1 und 2 ist er dafür verantwortlich, eine Schnittstelle für den Modellentwickler zur Verfügung zu stellen und dafür, dass Instanzen von bzgl. dieser Schnittstelle korrekt umgesetzten Szenariomodellen von der Simulationsanwendung ausführbar sind.

Modellentwickler: Dem *Modellentwickler* kommt die wohl größte Bedeutung zu [Ferstl & Sinz, 2013], da dieser dafür verantwortlich ist, deklaratives Domänenwissen in, dem Modellzweck entsprechende, ausführbare Modelle zu überführen. Dafür benötigt er neben allgemeinen Kenntnissen zur programmatischen Umsetzung von Strukturen und Abläufen zusätzlich Wissen bzgl. der vom *Simulationsentwickler* zur Verfügung gestellten Schnittstelle. Der *Modellentwickler* ist dabei in der Verantwortung, eine Vielzahl von Parametern vorzudefinieren, wie das Verhalten von Akteuren, Eigenschaften von Elementen, Umgebungsvariablen, Interaktions- und Kommunikationsregeln und Standardzustände (vgl. Abschnitt 7.2.4, insb. Abbildung 24). Der *Modellentwickler* stellt somit eine Menge möglicher Bestandteile einer Szenarioinstanz zur Verfügung. Zusätzlich trägt der *Modellentwickler* die Verantwortung für die Sicherstellung der korrekten Funktion und der Validität der von ihm erstellten Modelle.

Tester: Für die hier betrachtete Ausrichtung auf die Durchführung szenariobasierter Tests ist eine weitere, in den zuvor genannten Arbeiten nicht abgebildete Rollendefinition notwendig: der *Tester*. Diese

Rolle kann je nach aktueller Ausgestaltung und Phase des Entwicklungsprozesses auch durch dieselbe Person eingenommen werden wie die Rolle *System-/Komponenten-Entwickler* (vgl. Abbildung 42). Der *Tester* ist der eigentliche Nutzer der Simulationsanwendung: Er nutzt und parametriert die Modelle des Modellentwicklers, um konkrete Testfälle abzubilden, durchzuführen und auszuwerten. Der *Tester* benötigt dabei weder prozedurales noch deklaratives Detailwissen.

Die zwischen den Rollen *Modellentwickler* und *Tester* geschaffene Trennung der Verantwortlichkeiten ist ein essenzieller Schritt zur Erreichung der gesetzten Ziele. Obwohl diskrete Simulationssoftware im Allgemeinen effizienter und einfacher nutzbar geworden ist, gilt dies nicht direkt in gleichem Maße für Modellierungsaufgaben [Pidd, 1996]. Was dazu führt, dass, wenn der *Tester*/*Entwickler* selbst für eine spezifische Testreihe zunächst ein passendes Modell erstellen muss, in der Regel viel Wissen benötigt wird, das über seinen Fachbereich hinausgeht. Die Folgen können u. a. stark steigender Zeitaufwand aufgrund nötiger Einarbeitungen und nicht aussagekräftige Testreihen durch fehlerhafte oder unvollständige Modelle sein.

13.2 ANFORDERUNGEN AN DIE SOFTWARETECHNISCHE LÖSUNG

Die lösungsbezogenen Anforderungen sind weitestgehend in Einklang mit den Vorgaben der Norm [ISO/IEC/IEEE, 29148:2018] formuliert. Da diese keine konkreten Vorgaben zu einer einheitlichen Syntax von sprachlichen Anforderungsformulierungen macht, wird darüber hinaus auf die Anforderungsschablone für funktionale Anforderungen ohne zusätzliche Bedingung von Pohl & Rupp zurückgegriffen. In dieser werden die vier Modalverben *muss*, *sollte*, *wird* und *kann* zur gegenseitig ausschließenden Verwendung vorgeschlagen, um die Verbindlichkeit der jeweiligen Anforderung auszudrücken. Koelsch argumentiert, dass nur das Modalverb *shall (*dt.: muss) eine Daseinsberechtigung zur Formulierung von Anforderungen hat, da die anderen Verben keine unbedingte Erforderlichkeit ausdrücken und somit keine Anforderung im eigentlichen Sinne darstellen [Koelsch, 2016]. Die zur Formulierung der nachfolgend aufgelisteten Anforderungen verwendete Schablone wurde entsprechend angepasst und ist in Abbildung 43 dargestellt.

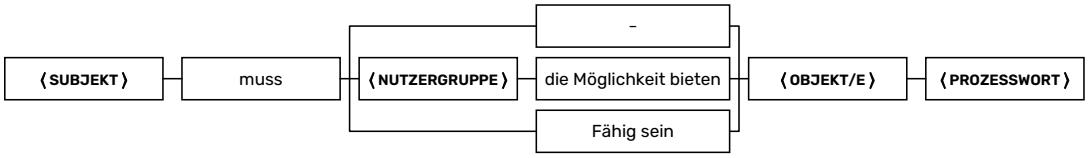


Abbildung 43: Angepasste Variante der Anforderungsschablone für funktionale Anforderungen ohne zusätzliche Bedingung nach [Pohl & Rupp, 2021]. Die Menge der Modalverben wurde gemäß [Koelsch, 2016] auf *muss* reduziert. Bei der Verwendung der Schablone sind Begriffe, die in ⟨WINKELKLAMMERN⟩ dargestellt sind, Platzhalter und entsprechend der zu formulierenden Anforderung zu ersetzen.

LA11 Szenarioinstanzen müssen mit unterschiedlichen, zur Erfüllung des konkreten Modellzwecks nötigen Auflösungen erstellt werden können.

Im Rahmen des angestrebten ganzheitlichen, entwicklungsbegleitenden Einsatzes (vgl. Abschnitt 7.1.2) stellt jede Kombination von zu testender Komponente und aktueller Entwicklungsprozessphase andere Anforderungen an das Szenariomodell. Um diesen vielfältigen Modellzwecken (vgl. Definition von *Modell* in Abschnitt 7.3.1) gerecht werden zu können, muss das Szenariomodell flexibel bzgl. des Detailgrads[60] und des Umfangs der Abbildung der Realität sein. Der IEEE-Standard 24765 definiert *test-*

[60] Unter anderem in [Tolk, 2012a] wird statt „Detailgrad“ der Begriff „Auflösung“ (engl.: resolution) genutzt.

ing als „[…] activity in which […] or component is executed under specified condition.“[61] und umfasst die sich verändernden Anforderungen an den Test ebenfalls. Ein allumfassendes Modell ist dabei keine valide Möglichkeit, weil (1) das Modell zwar alles für den spezifischen Test Nötige abbilden muss, gleichzeitig aber alles, was darüber hinausgeht, die Komplexität unnötig erhöht und somit die Nutzbarkeit verringert und die Fehleranfälligkeit erhöht [Tolk, 2012a] und (2) gerade in frühen Entwicklungsphasen hochaufgelöste Modelle nicht geeignet sind, um rudimentäre frühe Entwicklungsstände der sich in Entwicklung befindenden Komponente sinnvoll zu integrieren (vgl. Abschnitt 7.3.4).

LA12 Szenariomodelle müssen ontologische Verantwortungshierarchien abbilden.

Im Kontext automatisierter Fahrsysteme ist es nötig, sowohl Systeme als auch Teilsysteme, Komponenten und kleinteilige Softwareeinheiten testen zu können (vgl. Abschnitt 6.3). Diese sind oft in einer Verantwortungshierarchie organisiert, um ein effizientes Funktionieren des Gesamtsystems zu gewährleisten [Bossel, 1994]. Um das Umfeld der zu testenden Einheit im nötigen Detailgrad abbilden zu können (vgl. Anforderung LA01), ist daher ein hierarchischer Aufbau im Sinne einer Ontologie der Modelle und Modellinstanzen notwendig. (vgl. [Tolk, 2012a])

LA13 Szenarioinstanzen müssen einheitlich auf Basis verschiedener Szenariomodelle spezifiziert werden können.

Aus der Anforderung LA01 ergibt sich die Notwendigkeit, dass zur Spezifikation von Szenarioinstanzen verschiedene Szenariomodelle genutzt werden können. Um Teilziel 3 nicht zu verletzen, muss das Vorgehen zur Spezifikation einer Instanz eines konkreten Modells dabei unabhängig vom gewählten Modell einheitlich sein. Des Weiteren sind Modelle Wissensartefakte, und als solche ermöglichen sie Wissenschaftlern und Ingenieuren, bestehendes Wissen wiederzuverwenden. Die Erstellung von Modellen geht mit erheblichem Aufwand einher, und eine Wiederverwendung dieser sollte daher auch aufgrund potenzieller Kosten- und Arbeitsersparnisse angestrebt werden. [Balci et al., 2017]

LA14 Szenariomodelle müssen erweiterbar sein.

Um der Anforderung nach Flexibilität gerecht zu werden, gleichzeitig aber nicht den Arbeitsaufwand damit zu steigern, dass für jeden Modellzweck ein Szenariomodell von Grund auf neu erstellt werden muss, müssen vorhandene Modelle erweiterbar sein. Das erweiterte Modell muss anschließend für die Spezifikation von Szenarioinstanzen genutzt werden können.

LA15 Szenarioinstanzen müssen dem Tester die Möglichkeit bieten, den Variablen der referenzierten Modellelemente konkrete Werte zuweisen zu können.

Damit eine Überführung von konkreten Szenarien (vgl. Abschnitt 7.4.2) in Szenarioinstanzen möglich ist, müssen die Variablen der Modellelemente in der Spezifikation der Szenarioinstanz mit konkreten Werten belegt werden.

LA16 Szenarioinstanzen müssen unabhängig von der ausführenden Simulationsanwendung persistiert werden können.

Szenariobasierte Testprozesse lassen sich nur dann sinnvoll in Entwicklungs- und V&V-Prozesse integrieren, wenn die Szenarien mehrfach ausführbar, austauschbar und katalogisierbar sind. Der erste und essenzielle Schritt dahin ist, dass Szenarioinstanzen unabhängig von der Simulationsanwendung persistent gespeichert und übertragen werden können.

[61] Auslassung;

LA21 Die Simulationsanwendung muss sich vollständig aus einer Szenarioinstanz konfigurieren lassen und einen entsprechenden Simulationslauf durchführen können.

Die sogenannte Wiederverwendbarkeit der Konfiguration im Kontext von Simulationen beschreibt die Tatsache, dass Anwendungen eine wiederverwendbare Möglichkeit bieten, Simulationsszenarien zu konfigurieren. Das bedeutet, dass die Konfiguration außerhalb der Simulationsanwendung selbst liegend betrachtet wird. „Außerhalb" bedeutet in diesem Zusammenhang, dass beliebige Konfigurationen wiederverwendet werden können, ohne die Implementierung der Simulationsanwendung verändern zu müssen. [Krammer et al., 2015]

In wissenschaftlichen Veröffentlichungen zu Anforderungen an Simulationsanwendungen wird zudem häufig eine einfache Nutzbarkeit genannt, worunter eine zentrale Orchestrierung und eine einfache Ausführung einzuordnen sind (vgl. [Gütlein, 2023; Oppelt et al., 2014]). Der Erfüllung dieser Eigenschaften ist die geforderte vollständige Simulationskonfiguration aus einer Szenarioinstanz zuträglich.

LA22 Die Simulationsanwendung muss die Durchführung komponentenbasierter Verkehrssimulationsläufe ermöglichen.

Eine der wichtigsten vorab zu bestimmenden Eigenschaften einer Simulation ist ihre Granularität [Jesenski et al., 2019]. Um die Sicherheit eines System-of-Systems sowie seiner Bestandteile (vgl. Abschnitt 6.3) simulativ testen zu können, muss es möglich sein, das Verhalten von Verkehrsteilnehmern einschließlich das ihrer Teilsysteme und Komponenten abzubilden. Außerdem muss es möglich sein, die einzelnen System-, Teilsystem- und Komponentenmodelle auszutauschen. Es ist somit klar, dass eine submikroskopische Verkehrssimulation, auch komponentenbasierte Verkehrssimulation genannt (vgl. Abschnitt 7.3.3.1), umzusetzen ist, da eine solche in der Lage ist, die interne Systemstruktur des Fahrzeugs abzubilden [Jesenski et al., 2019].

LA23 Die Simulationsanwendung muss stochastische Einflüsse auf das Verhalten der Akteure während eines Simulationslaufs ermöglichen

Stochastische Einflüsse müssen für einige Anwendungsfälle explizit berücksichtigt werden, z. B. durch die Angabe von Übergangswahrscheinlichkeiten zwischen Systemzuständen oder zufälligen Schwankungen von Eingangsgrößen. Stochastische Modelle liefern daher für jeden Simulationslauf unterschiedliche Ergebnisse. Eine große Zahl von Simulationen (Monte-Carlo-Simulation) kann dann einen Überblick darüber verschaffen, welche statistische Verteilung von Verhalten zu erwarten ist, wo die Mittelwerte liegen und mit welcher Streuung zu rechnen ist.

Auch einige Ansätze zur Identifizierung von seltenen Zuständen im Zustandsraum des Simulationsmodells, um diese als Basis zukünftiger Testszenarien zu nutzen, sehen den Einsatz von Simulationsläufen in einer Art Feedbackschleife vor (*Importance Splitting*). Stochastische Entwicklungen würden zusätzlich auch ein solches Erkunden des Zustandsraums ermöglichen.

LA24 Die Simulationsanwendung muss in der Lage sein, deterministisches Verhalten von Akteuren in einem Simulationslauf abzubilden.

Viele der bestehenden Ansätze zur Spezifikation von Szenarien setzen auf eine vollständige Beschreibung des Verhaltens der Elemente in der relevanten Umwelt des zu testenden Systems (vgl. u. a. [Austel et al., 2024; Damm, 2018; Bach et al., 2016]). Bei einer simulativen Ausführung eines solchen Szenarios sind zufallsbeeinflusste Veränderungen und Einflüsse ausgeschlossen und der Simulationslauf somit deterministisch.

Die zu entwickelnde Simulationsanwendung soll ebenfalls die Möglichkeit bieten, deterministische Abläufe abzubilden. Nötig ist dies für Anwendungsfälle, in denen ein Szenario oder ein Teil des Szenarios auf aufgezeichneten historischen Daten basieren soll, um z. B. reale Situationen nachzustellen und simulativ zu erproben, wie sich die zu testende Komponente in dieser verhält. Auch für Anwendungsfälle, in denen die Reproduzierbarkeit der Ergebnisse relevant ist, ist die Abwesenheit von stochastischen Einflüssen notwendig.

LA25 Die Simulationsanwendung muss die Möglichkeit bieten, einen Simulationslauf nach dessen Ende bezüglich des Erfolgs oder Misserfolgs des Tests auszuwerten.

Da das Ziel dieser Arbeit ist, simulative Tests (vgl. Abschnitt 7.2.1) als Teil von entwicklungsbegleitender Validierung und Verifizierung softwarebasierter Fahrfunktionen zu ermöglichen, muss es möglich sein, die Simulationsläufe auf mögliche aufgetretene Fehlerwirkungen hin zu untersuchen (vgl. Definition von *testing* in [ISO/IEC/IEEE, 24765:2017]). Bei Abwesenheit von Fehlerwirkungen gilt der Test als bestanden und bei dem Vorhandensein von Fehlerwirkungen als fehlgeschlagen. Diese Auswertung soll zwar zunächst nicht von der Simulationsanwendung selbst durchgeführt werden, aber es muss ermöglicht werden, die Zustandstrajektorie des gesamten Simulationslaufs im Nachgang extern automatisiert auf Fehlerzustände zu untersuchen (vgl. [Meyer, 2008]). Eine solche Auswertung kann u. a. durch eine automatisierte Prüfung der Zustandstrajektorie auf das Einhalten zuvor festgelegter Grenzwerte bestimmter Zustandsvariablen erfolgen.

LA26 Die Simulationsanwendung muss die Beobachtung des Modellzustands während eines Simulationslaufs ermöglichen.

Diese Anforderung erweitert die Anforderung A12 um die notwendige Fähigkeit der Simulationsanwendung, den Zustand und die Zustandstrajektorie des Simulationslaufs zur Laufzeit zum Zwecke der Weiterverarbeitung zur Verfügung zu stellen. Dies ermöglicht u. a. den Abbruch eines Simulationslaufs, sobald eine vordefinierte Bedingung erfüllt wurde, den Anschluss externer Werkzeuge wie einer grafischen Monitoranwendung und die Beobachtung der Zustandsentwicklung zu Debugging-Zwecken.

LA31 Das Simulationsmodell muss dem Tester während der Szenariospezifikation die Möglichkeit bieten, auf vordefinierte Teilmodelle zurückzugreifen.

Als einen von vier Gründen für eine hohe Komplexität von Simulationsmodellen wird in [Chwif et al., 2000] die Möglichkeit genannt, dass der Ersteller des Modells das zu modellierende System und seine Umwelt nicht vollumfänglich verstanden hat. Die in Abschnitt 13.1 beschriebene angestrebte Trennung der Rollen des Modellentwicklers und des Testers hält bereits einen Großteil der Komplexität der Modellierung von der Person, die die Szenarioinstanz erstellt, fern (vgl. Anforderung A15). Da der Tester vor allem im wissenschaftlichen Kontext mit hoher Wahrscheinlichkeit der Entwickler der zu testenden Komponente ist und die Entwicklung dessen kein vollumfängliches Wissen über das zukünftige Gesamtsystem und seine Umgebung voraussetzt, müssen vordefinierte Teilmodelle zur Spezifikation von Szenarien verfügbar sein. So könnten z. B. Modelle von Schiffen inkl. aller nötigen Komponenten vordefiniert und allen wesentlichen Variablen vorbelegt durch den Modellentwickler zur Verfügung gestellt werden (vgl. [Ramos & Piera, 1999]).

LA32 Der Szenariospezifikationsprozess muss dem Tester die Möglichkeit bieten, Szenarioinstanzen zu spezifizieren, ohne dass Kenntnisse über prozedurale Details des Modells und der Simulation

und detaillierte deklarative Kenntnisse bzgl. Wirkzusammenhängen der modellierten Domäne vorhanden sein müssen.

In Abschnitt 13.1 wurde die Rolle des Testers von der des Modellentwicklers getrennt definiert, um die Komplexität der Modellstrukturen und Simulationsabläufe von dem tatsächlichen Anwender der Simulationsanwendung fernzuhalten. Diese Anforderung zielt auf die Umsetzung dieser Trennung ab, indem gefordert wird, dass die Erstellung von ausführbaren Szenarioinstanzen möglich sein muss, ohne Wissen aus den Aufgabenbereichen des Modell- und des Simulationsentwicklers einbringen zu müssen.

LA33 Der Modellierungsprozess muss dem Modellentwickler die Möglichkeit bieten, Modelle zu erstellen, ohne dass Kenntnisse über prozedurale Details der Simulation vorhanden sein müssen.

In Abschnitt 13.1 wurde die Rolle des Modellentwicklers von der Rolle des Simulationsentwicklers vollständig getrennt definiert, um die prozedurale Komplexität der zugrundeliegenden Implementierungsdetails der Simulationsanwendung von dem Modellierer fernzuhalten. Diese Anforderung zielt auf die Umsetzung dieser Trennung ab, indem gefordert wird, dass die Erstellung von Modellen möglich sein muss, ohne prozedurales Detailwissen aus den Aufgabenbereichen des Simulationsentwicklers einbringen zu müssen (vgl. Abbildung 44).

LA34 Die Szenariospezifikations- und Szenariosimulationsprozesse müssen dem Tester die Möglichkeit bieten, Szenarioinstanzen zu erstellen und zu simulieren, ohne dass manuelle Transformationen der Zwischenergebnisse nötig sind.

Der szenariobasierte Testprozess umfasst in der Regel mindestens drei Phasen (vgl. Abbildung 2): die Identifizierung relevanter Szenarien, die Modellierung dieser Szenarien sowie die anschließende simulative Ausführung und Auswertung der Testfälle. Diese Prozessschritte werden oft unabhängig voneinander durch mehrere Personen unter Verwendung unterschiedlicher Konzepte, Werkzeuge, Dateiformate etc. durchgeführt. Die aus den einzelnen Schritten resultierenden Informationen sind an ein bestimmtes Format gebunden. Sie können unter anderem sprachlich formuliert, in strukturierter textbasierter Form gespeichert oder als flüchtige Objektinstanz vorliegen. Wenn vor der weiteren Verarbeitung der Informationen ein Übergang zu einem anderen Format erforderlich ist, bedeutet dies mindestens einen Medienbruch, oft sogar einen Paradigmenwechsel. Der Übergang von einem Format oder einem Paradigma zu einem anderen kann zu Informationsverlusten oder -verzerrungen führen (vgl. Abschnitt 7.4). Daher sollen manuelle Modelltransformationen während des Prozesses vermieden werden. Ein Beispiel für einen Prozess, der solche, hier ungewollten, Transformationen enthält, ist im Standard der *Simulation Interoperability Standards Organization* (SISO) zur *Military Scenario Definition Language* (MSDL) [SISO, 007:2008] in Abbildung 1 dargestellt: Der Gesamtprozess umfasst die drei Artefakte *Planning Scenario*, *MSDL Scenario* und *Execution Scenario*, wobei die letzten beiden jeweils durch Transformation aus dem vorhergehenden erzeugt werden müssen.

LA41 Der Szenariospezifikationsprozess muss dem Tester die Möglichkeit bieten, eine Repräsentation der zu testenden Einheit auf allen Hierarchieebenen in die Szenarioinstanz zu integrieren.

Für den Einsatz während des gesamten Entwicklungsprozesses (vgl. Abschnitt 7.1.2) ist es nötig, die zu testende Einheit auf der Hierarchieebene der Modellinstanz zu integrieren, die der Ebene der Verantwortungshierarchie des Systems oder Systemverbunds (vgl. Abschnitt 6.3) des realen Einsatzes ent-

spricht. Bei umfassender Erfüllung dieser Anforderung ergibt sich auch die Möglichkeit, Szenarioinstanzen für alle in der untersten Zeile von Tabelle 5 genannten Testarten zu erstellen.

LA51 Die Simulationsanwendung muss fähig sein, bidirektional mit allen relevanten Arten externen Komponenten zu kommunizieren.

Die Anforderungen LA08 und LA18 bedingen, dass die Simulationsanwendung einen bidirektionalen Datenaustausch entsprechend der Konfiguration zu Beginn des Simulationslaufs initiiert und für die Zeit des Simulationslaufs aufrechterhält.

13.3 ZUSAMMENFASSUNG

Die in Abschnitt 13.2 formulierten lösungsorientierten Anforderungen decken alle in den zuvor formulierten gestaltungsorientierten Teilzielen inhaltlich enthaltenen Aspekte ab und konkretisieren diese in Hinblick auf die Überprüfbarkeit der Erreichung dieser. Gleichzeitig adressieren sie direkt die als Ergebnis der explorativen Phase aufgestellten Handlungsempfehlungen und stellen eine präzisierende Dekomposition und Erweiterung der übergreifenden Anforderungen aus Abschnitt 9 dar. Die inhaltliche Zuordnung der Anforderungen zu den Teilzielen ist in Abbildung 44 dargestellt. Die Anforderungen stellen somit, im Sinne eines systematischen Softwareentwicklungsprozesses (vgl. Abbildung 4), die Basis für den nachfolgenden praktischen Teil dar. Der Anspruch der entwickelten Konzepte und der prototypischen Umsetzung dieser ist es somit, eben diese Anforderungen vollständig zu erfüllen. Die Anforderungen entsprechen somit den kleinsten separiert überprüfbaren Einheiten, deren Erfüllung später zur Evaluierung des praktischen Teils dieser Arbeit untersucht wird (vgl. Abschnitt 16.1). Basierend auf der konsequenten hierarchischen Herleitung der Ziele und Anforderungen kann davon ausgegangen werden, dass bei Erfüllung aller Anforderungen, die einem Ziel zugeordnet sind, auch das jeweilige Ziel erreicht wurde.

14 KONSTRUKTIVER LÖSUNGSENTWURF

Ingenieursmäßiges Vorgehen bedeutet, systematisch, organisiert und geplant unter Einsatz etablierter Methoden vorzugehen [Herrmann, 2022], um durchweg hochwertige Ergebnisse zu erzielen. Dies steht im Gegensatz zu flexibleren Ansätzen, welche schnelle Teilergebnisse priorisieren und dabei die Qualität dieser in Hinblick auf nötige Anpassungen auf dem Weg zum Endergebnis wenig bis gar nicht

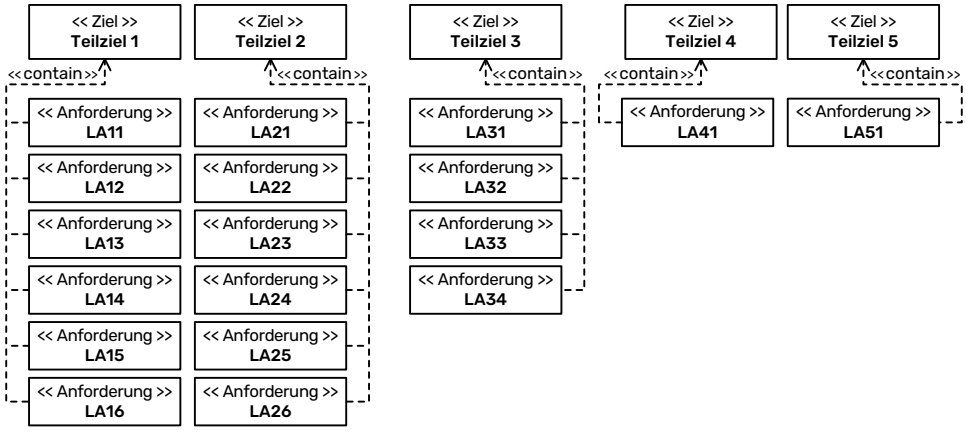


Abbildung 44: Anforderungsdiagramm zur Darstellung der identifizierten lösungsorientierten Anforderungen, die an die zur Erreichung der gestaltungsorientierten Ziele entwickelten technischen Konzepte gestellt werden.

betrachten. Zu diesem Zweck wurden die Teilkonzepte zur Erreichung der gestellten Ziele vorab unter Beachtung ihrer Abhängigkeiten untereinander ausgearbeitet. In dieser Phase wurden verschiedene Entwurfsalternativen unterschiedlichen Detailgrads bereits bewertet, um idealerweise die optimale Lösung schon in einer frühen Phase des Gesamtprozesses zu ermitteln. Nachfolgend werden als Ergebnis dieser Arbeit davon lediglich die Konzepte vorgestellt, die am Ende dieses Prozesses als Ergebnis hervorgegangen sind. Zunächst werden in Abschnitt 14.1 die Lösungskonzepte auf einer technologieunabhängigen und abstrakten Betrachtungsebene mit Bezug zu den einzelnen in Abschnitt 12 definierten gestaltungsorientierten Zielen vorgestellt. Anschließend werden, den Teilkonzepten nahe, bereits existierende Ansätze vorgestellt und ein Abgleich dieser mit der jeweiligen Zielsetzung durchgeführt, um die Notwendigkeit und Relevanz der erarbeiteten Konzepte zu betonen. Dabei ist der entsprechende Abschnitt 14.2 in die drei zentralen konzeptionellen Teilbereiche *hierarchische Szenariomodellierung*, *verteilte Simulation* und *Modelltransformation* gegliedert. Die konstruktive Entwurfsphase abschließend werden in Abschnitt 14.3 die relevantesten Konzeptteile detaillierter vorgestellt und gestalterische Entscheidungen durch inhaltliche Schlussfolgerungen begründet.

14.1 LÖSUNGSKONZEPTE

Die Grundlage für das nachfolgend vorgestellte übergreifend allgemeine Lösungskonzept bilden die in Abschnitt 13 vorgestellten lösungsbezogenen Anforderungen. Aufgrund dessen, dass diese jeweils einem der fünf Teilziele Z1 bis Z5 zugeordnet werden können (vgl. Abbildung 44), adressieren die Unterabschnitte 14.1.2 bis 14.1.7 jeweils eines dieser Teilziele und stellen das Konzept zur Erreichung dessen vor. Zunächst werden aber in Unterabschnitt 14.1.1 für alle Teilbereiche allgemeingültige konzeptionelle Entscheidungen begründet vorgestellt. Unterabschnitt 14.1.8 verknüpft anschließend die vorgestellten Teilkonzepte zu einem homogenen Gesamtbild.

14.1.1 AGENTENBASIERTE SUBMIKROSKOPISCHE VERKEHRSSIMULATION

Das übergeordnete gestaltungsorientierte Ziel *Z0* fordert eine Simulationsanwendung, die für Testläufe maritimer Fahrsysteme eingesetzt werden kann. Die dafür nötige submikroskopische Betrachtungsebene ist ebenfalls bereits durch die Formulierung des Ziels vorgegeben. Da im Rahmen von Simulationen dieser Betrachtungsebene, wie auch für mikroskopische, im Gegensatz zu makroskopischen (vgl. Abschnitt 1), die Möglichkeit gegeben sein muss, den aktuellen Zustand jedes einzelnen simulierten Verkehrsteilnehmers zu beobachten, ist es naheliegend, sowohl die Strukturen der zugrundeliegenden Modelle als auch die Abläufe der Simulationsausführung im Sinne eines agentenbasierten Systems (vgl. Abschnitt 7.3.3.2) zu realisieren (vgl. [Ghadai et al., 2016]). Da die meisten relevanten Testszenarien für Fahrfunktionen mehr als einen Verkehrsteilnehmer umfassen, reicht die Modellierung eines einzelnen Agenten nicht aus. Zusammen mit der Entscheidung für eine agentenbasierte Struktur ergibt sich daher, dass die Simulationsanwendung eine *Multiagentensimulation* (MAS) sein muss.

Die Verwendung multiagentenbasierter Modelle ist ein logischer Schritt für die realistische Abbildung von Seeverkehr, da dieser bzw. Verkehr im Allgemeinen ein komplexes, selbstorganisierendes Gesamtsystem aus einer Vielzahl von autonomen Einheiten ist. In anderen Verkehrsbereichen, wie dem Straßenverkehr und dem Verhalten von Fußgängern, wurden ebenfalls bereits gute Ergebnisse erzielt. Simulationsläufe mit agentenbasierten Modellen zeigen dabei sowohl auf der Ebene einzelner Verkehrsteilnehmer als auch auf der Ebene des Verkehrs als Gesamtes Vorteile. Auf der Ebene der

Agenten kann das individuelle Verhalten realistisch sein und auf der Ebene des Verkehrs können sich statistische Eigenschaften zeigen, die man von realem Verkehr erwarten würde. Zusätzlich haben Multi-Agenten-Ansätze das Potenzial, Interaktionen, emergentes Verhalten und Unsicherheiten aus sich heraus darzustellen. Viele bestehende Modelle zur Simulation von Schiffsverkehr bieten diese Möglichkeiten nicht [Xiao et al., 2013]. Auch mit Blick auf den geplanten Einsatzzweck zeigt sich eine Multi-Agenten Architektur in zweierlei Hinsicht als sinnvoll: (1) Externe zu testende Systeme können intuitiver angebunden werden, da sie sich nicht stark von der Definition eines Agenten unterscheiden [Ghadai et al., 2016]; (2) die geringere Abhängigkeit von externer Ablaufplanung und -steuerung ermöglicht im Gegensatz zu Ansätzen wie der *diskreten ereignisbasierten Simulation* (vgl. [Banks et al., 2005]) ein quasi-autonomes Handeln der einzelnen Agenten [Martinez et al., 2024].

Nach Ghadai et al. [2016] müssen für den sinnvollen Einsatz von Verkehrssimulationen mindestens die drei folgenden Bestandteile modelliert werden können: Fahrzeuge, Fahrer und Umgebung. Ein einzelnes Fahrzeug wird hier als Agent angenommen, was auch der Definition eines Akteurs als Element eines Szenarios entspricht (vgl. Abschnitt 7.4). Das Verhalten des Agenten kann sowohl durch ein einfaches Systemmodell (SM) im Sinne einer Übertragungsfunktion als auch durch die Abbildung einer hierarchischen Wirkstruktur modelliert werden (vgl. Abschnitt 7.3). Der Fahrer kann im zweiten Fall als Teil der Wirkstruktur abgebildet werden, denn das ist er in der Realität ebenfalls. Die verhaltensbeschreibenden Teilsystem- und Komponentenmodelle dieser Wirkstruktur müssen dabei sowohl ganz oder teilweise seriell als auch parallel durchlaufen werden können, um sowohl voneinander unabhängig wirkende Teile wie eine Lichtautomatik als auch hierarchisch angeordnete wie eine Motorsteuerung, die den Eingaben eines Autopiloten folgt, abbilden zu können. Um einheitliche Abläufe zu gewährleisten, sind die Agenten selbst so umzusetzen, dass der Ablauf stets der folgenden Reihenfolge entspricht: (1) Lesen von Werten aus der Umgebung, (2) Überführung der Eingangsgrößen in eine interne Zustandshaltung, (3) Durchlaufen der internen Wirkstruktur. Der letzte Schritt kann wiederum zu Veränderungen der Inhalte des internen Zustands führen, welche bei Bedarf zentral vom Agenten als Ausgangsgrößen an die simulierte Umwelt weitergegeben werden.

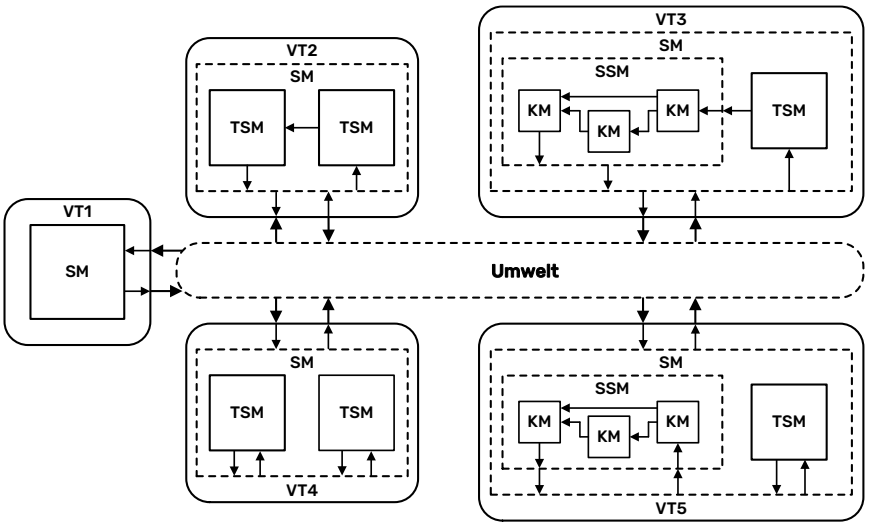


Abbildung 45: Individuelle Verkehrsteilnehmer als Agenten einer submikroskopischen Verkehrssimulation. Die modellierte Wirkstruktur eines Verkehrsteilnehmers (VT) kann unterschiedliche Ausprägungen bezüglich des Detailgrads annehmen. Systemmodelle (SM) können durch eine Menge von Teilsystemmodellen (TSM) verfeinert werden, welche wiederum durch eine Menge von Komponentenmodellen (KM) abgebildet werden können.

Eine solche Multi-Agenten-Verkehrsumgebung inklusive aller Wirkbeziehungen ist beispielhaft in Abbildung 45 dargestellt. Die fünf dargestellten Agenten repräsentieren Verkehrsteilnehmer (VT1–VT5) und stellen zudem beispielhaft die zuvor beschriebenen möglichen Ausprägungen der internen Wirkstruktur eines Agenten dar.

14.1.2 KOMPONENTENBASIERTES SZENARIO-METAMODELL

Zur Erreichung des ersten gestaltungsorientierten Teilziels Z1 müssen zwei Aspekte sichergestellt werden: Flexibilität und Wiederverwendbarkeit der Szenariomodelle in einem Umfang, der den Einsatz während des gesamten Entwicklungsprozesses verschiedenster maritimer Fahrsysteme, Teilsysteme und Komponenten ermöglicht. Die Grundlage dafür ist eine einheitliche Modellierungsbasis, welche allen Modellen gemeinsame Strukturen vereinheitlichend modellierend vorgibt.

Ein solcher gemeinsamer struktureller Unterbau vereinfacht zudem die Arbeit des Modellentwicklers und beseitigt Unsicherheiten, da (1) eine gewisse Wirkstruktur bereits vorgegeben ist, welche dem verwendenden Modellentwickler eine gewisse Denkweise „aufzwingt“, und (2) die allgemeinen Wirkstrukturen einer szenariobasierten Verkehrssimulation vorgegeben sind und wiederverwendet werden können, statt diese wiederholt bei der Entwicklung eines Modells aufs Neue identifizieren und umsetzen zu müssen. Beide Punkte zusammen gehen effektiv die u. a. von Krajzewicz [2010] identifizierten Probleme der fehlenden Wiederverwendbarkeit und unnötig wiederholten Entwicklungsarbeit im wissenschaftlichen Einsatz von Verkehrssimulationen an. Im Kontrast zu der in [Krajzewicz, 2010] vorgestellten sowie anderen gängigen Lösungen bleibt die wiederverwendbare Basis der hier vorgestellten Lösung dabei auf einer abstrakteren Betrachtungsebene, sodass die Anwendungsmöglichkeiten nicht durch starke Spezialisierung auf einen Teilbereich von Fragestellungen oder eine einzelne Verkehrsdomäne beschränkt sind.

Eine bewährte Methode, um die erforderliche Flexibilität zu erreichen, ist, das Modell selbst zu modellieren. Ein solches Modell des Modells wird als Metamodell bezeichnet (vgl. Abschnitt 7.4.3). Das Metamodell befasst sich mit den potenziellen Strukturen, die sich im Modell manifestieren lassen. Metamodelle können somit direkt auf die spezifischen Anforderungen bestimmter Anwendungsfallmengen ausgerichtet werden. Ein bekanntes Beispiel für ein Metamodell ist die *Unified Modeling Language* (UML)[62] respektive ihre Spezifikation. UML ist dabei eng verbunden mit dem *Meta Object Facility* (MOF)[63]-Standard zur Erstellung solcher Metamodelle. Es handelt sich dabei um eine Untergruppe von UML [OMG, UML:2.5.1]. Genauer gesagt haben UML und MOF einen gemeinsamen Kern, der wiederum Teil des UML-Standards ist. Die Schicht M3 umfasst die Spezifikation der Modellierungssprache, die zur Darstellung der in M2 befindlichen Metamodelle verwendet wird, und stellt somit ein Metamodell des Metamodells dar. Das UML-Metamodell stellt dabei die Konzepte zur Verfügung, die für die Erstellung von UML-Modellen nötig sind. Für UML-Klassendiagramme sind das u. a. Klassen, Attribute und Instanzen. Mithilfe dieser Konzepte können Modellierer die Wirklichkeit in Form eines UML-Klassendiagramms abbilden. Dabei sind die Elemente einer Schicht immer Instanzen der Konzepte der darüberliegenden Schicht (vgl. Abbildung 46).

Um die zuvor beschriebene nötige gemeinsame strukturelle Basis zu schaffen, wird ein Metamodell eingeführt. Im Gegensatz zu UML zielt das eingeführte Metamodell aber nicht auf breite allgemeine

[62] Object Management Group, *Unified Language Model*: `https://www.omg.org/uml/` [letzter Zugriff: 17.02.26]
[63] Object Management Group, *Meta Object Facility*: `https://www.omg.org/mof/` [letzter Zugriff: 17.02.26]

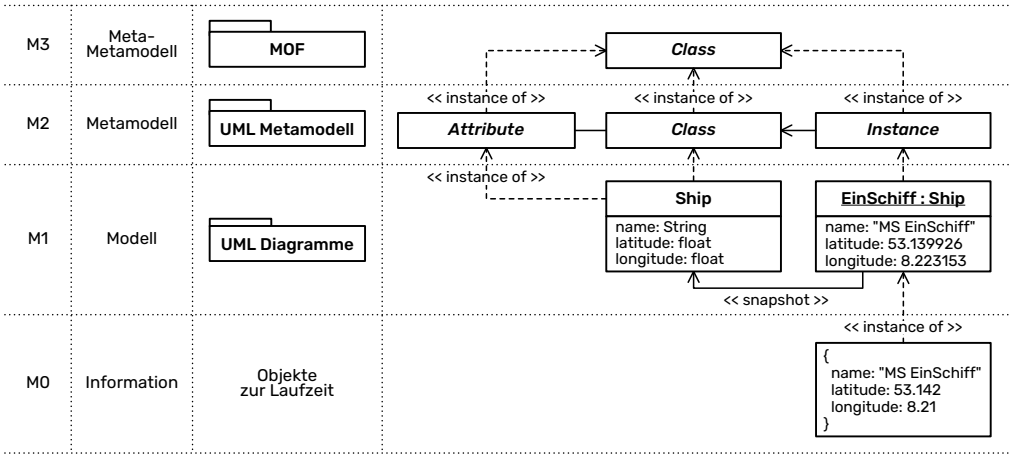


Abbildung 46: Das Konzept der Metamodellierung, wie es im Meta-Object-Facility (MOF)-Standard für modellgetriebene Entwicklung beschrieben wird, angereichert mit einem vereinfachten UML-basierten Minimalbeispiel.

Verwendbarkeit ab, sondern definiert Konzepte, die speziell auf die Modellierung von simulierbaren Szenarien ausgerichtet sind. Das Metamodell stellt somit im weitesten Sinne eine *domänenspezifische Modellierungssprache* (engl.: Domain Specific Modelling Language, DSML) dar. Die Verwendung von DSMLs verspricht im direkten Vergleich mit allgemeinen Metamodellen einige Vorteile, die sich mit den hier angestrebten Eigenschaften und Zielen decken, wie der Verschlankung des Modellierungsprozesses sowie der Förderung der Verständlichkeit und Integrität der erstellten Modelle [Frank, 2014].

Zum Zwecke einer breiten Einsetzbarkeit des Metamodells und einer Wiederverwendbarkeit von Teilmodellen muss das Metamodell eine für alle potenziellen Modelle allgemeine Struktur- und Verhaltensbeschreibung bieten. Wichtig dabei ist an dieser Stelle ein Fokus auf die inhaltlichen Aspekte der möglichen Modelle statt auf die prozeduralen [Ramos & Piera, 1999]. In Abschnitt 7.2.4 wurden die Elemente eines Szenarios, angelehnt an [Ulbrich et al., 2015] und [Steimle et al., 2021], hergeleitet und ausführlich beschrieben. Das Ergebnis davon wurde in Abbildung 24 strukturiert dargestellt. Aufgrund der Tatsache, dass im Zentrum der hier beschriebenen konstruktiven Lösung ebensolche Szenarien stehen, werden die dort dargestellten Elemente inklusive der Beziehungen zwischen ihnen als Grundlage für das Metamodell genutzt. Um dem definierten Einsatzzweck und den aufgestellten Anforderungen gerecht zu werden, wurden zusätzlich zwei Erweiterungen eingeführt:

Komponenten: In Abschnitt 7.3.1 wurde unter anderem der Unterschied zwischen systemerklärenden und verhaltensbeschreibenden Modellen aufgezeigt und in Abbildung 27 beispielhaft veranschaulicht. Aufgrund der Tatsache, dass automatisierte Schiffe Systemverbünde im Sinne eines System-of-Systems (SoS) darstellen und somit eine submikroskopische Betrachtung ermöglicht werden muss (vgl. Abschnitte 7.3.3.1 und 14.1.1), müssen sowohl Teilsysteme eines Akteurs als auch die Komponentenhierarchien der Teilsysteme abgebildet werden können (vgl. Abbildung 14). Um das Metamodell möglichst schlank zu halten, wird bei der Modellierung nicht zwischen System und Komponente unterschieden. Stattdessen kann ein System durch eine Komponente auf oberster Hierarchieebene abgebildet werden. Möglich ist das, da Systeme und Komponenten sich lediglich in ihrer internen Zusammensetzung unterscheiden, nicht aber in ihrer von außen sichtbaren Struktur oder ihren internen Abläufen: Beide erhalten Eingangsgrößen, geben diese ggf. an untergeordnete Komponenten weiter und erzeugen Ausgangsgrößen. Durch diese optionale Verschachtelung gleichartiger Elemente ist es ebenfalls möglich, verhaltensbeschreibende Modelle zu erstellen: Ein Akteur, welcher keine weitere Wirkstruktur in Form

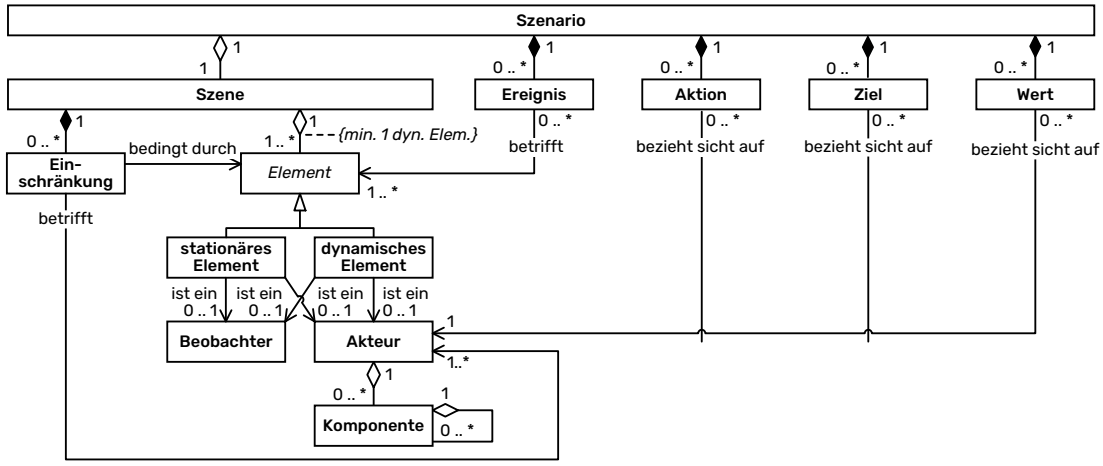


Abbildung 47: Inhalte eines Szenarios, erweitert um geschachtelte Komponenten innerhalb eines Akteurs, zur Abbildung der internen Wirkstrukturen von modernen Fahrzeugen (vgl. Abbildung 24)

von Komponenten besitzt und stattdessen die Ausgangssignale direkt aus den Eingangssignalen erzeugt und ausgibt, kann als verhaltensbeschreibendes Modell angesehen werden. Somit ist diese Struktur des Metamodells im Einklang mit den Anforderungen LA01 und LA02.

Einschränkungen: Jede Verkehrsart hat Regeln, die es im Normalfall einzuhalten gilt, damit ein geregelter und weitestgehend gefahrloser Ablauf gewährleistet werden kann. Um einen flexiblen Umgang mit diesen zu ermöglichen, wurde der gegebenen Struktur ein weiteres Element zur Abbildung von Einschränkungen hinzugefügt. Die Startszene eines Szenarios kann beliebig viele solcher Einschränkungen enthalten (d. h. auch keine). Eine Einschränkung betrifft immer bestimmte Arten von Elementen, wie einen bestimmten Schiffstyp. Einschränkungen, die durch ein bestimmtes Element bedingt sind, sind ebenso möglich wie solche, die eine globale Gültigkeit haben. Einschränkungen begrenzen dabei den validen Wertebereich bestimmter Zustandsvariablen der betroffenen Elemente.

Die um die beiden genannten Konzepte erweiterte grafische Darstellung der möglichen Inhalte eines Szenarios ist in Abbildung 47 dargestellt. Die abgebildeten Elemente und Beziehungen bilden das Metamodell, welches die geforderte gemeinsame Basis für die Erstellung von unterschiedlichen Modellen liefert. Das abgebildete Metamodell ist in der Lage, alle Bestandteile einer Verkehrssituation strukturell auf einer hohen Ebene abzubilden. So lassen sich zum Beispiel alle von Scholtes et al. [2021] beispielhaft für jede Ebene des dort vorgeschlagenen Modells (vgl. Abschnitt 14.2.1) zur Beschreibung von Verkehr aufgeführten Entitäten abbilden. Ausgenommen davon sind Eigenschaften wie *vereist* oder *nass*, da diese Teil des Zustands eines Elements sind und somit keine eigene Entität darstellen. Ebenso wird hier auf die Modellierung von Kommunikationskanälen oder digitalen Nachrichten verzichtet, da die Agenten eine solche Kommunikation selbstständig aushandeln sollten, um die Realität solcher Kommunikation besser abzubilden. Die Konzeptaspekte, die über die reine Strukturbeschreibung hinausgehen, wie die Integration von Eigenschaften von Elementen und der möglichen Kommunikation zwischen Elementen, werden später im weiteren Entwurfsverlauf im Detail aufgegriffen.

14.1.3 EINGEBETTETE MODELLIERUNGSHIERARCHIE

Aufgrund der kausalen Abhängigkeitsbeziehung des Umfangs und des Detailgrads von der individuellen Aufgabe, verkörpert durch das Ziel des konkreten zu simulierenden Testfalls (vgl. Abbildung 48), fordert das gestaltungsorientierte Teilziel 1 die Möglichkeit, Modelle erweitern zu können und dabei

(Teil-)Modelle möglichst flexibel wiederverwenden zu können. Zudem hat ein hoher Grad an Wiederverwendbarkeit das Potenzial, bedeutende Arbeits- und Kosteneinsparungen zu ermöglichen und somit dem prominenten Kritikpunkt des hohen Aufwands, der mit szenariobasierten simulativen Testläufen in Verbindung gebracht wurde, entgegenzuwirken: Modelle sind Wissensartefakte, und als solche erlaubt ihre Wiederverwendung, auf bereits erarbeitetes und ggf. erprobtes Wissen aufzubauen. Modelle manifestieren sich in der Regel als Software, die mit erheblichem Aufwand entwickelt und erprobt wurde [Balci et al., 2017].

Abbildung 48: Einfluss der Ziele eines Tests auf den Umfang und den Detailgrad des optimalen Modells

Die Aufteilung in mehrere Untermodelle und die Gestaltung der entsprechenden Schnittstellen sind relevante Aspekte bei der Betrachtung der Wiederverwendbarkeit von Modellkomponenten [Gütlein, 2023]. Eine solche Struktur muss modular und parametrierbar, aber trotzdem einheitlich und vor allem eindeutig sein. Die Basis dafür, dass Szenarien austauschbar sowie die Simulationsläufe und -ergebnisse trotz der angestrebten Flexibilität vergleichbar sind, schafft das im vorigen Abschnitt beschriebene Metamodell. Aufbauend auf diesem wurde für das eigentliche Modell eine hierarchische Struktur erarbeitet, welche die nötige Auftrennung in klar definierte Teilmodelle ermöglicht.

In ihrer Arbeit zur modellgetriebenen Entwicklung haben Atkinson & Kühne zusätzlich zur klassischen linguistischen Dimension eine weitere Dimension hierarchischer Modellierung eingeführt. Sie bezeichnen diese als ontologische Modellierungshierarchie und charakterisieren sie als die Beschreibung der in einem abgegrenzten thematischen Bereich vorhandenen Konzepte und ihrer Eigenschaften [Atkinson & Kühne, 2005, 2003]. Diese Idee wurde auch von anderen Arbeiten aufgegriffen und u. a. in [Henderson-Sellers et al., 2013] in angepasster Form umgesetzt. Diese neuen Schichten, die mit O_0 bis O_n identifiziert werden, unterteilen hier das Szenariomodell in drei konzeptuelle Ebenen und befinden sich daher auf der linguistischen Ebene L_1. Um die Wiederverwendbarkeit zu verbessern, hat sich die folgende Unterteilung als zielführend gezeigt: (O_2) ein domänenunabhängiges Metamodell für Verkehrsszenarien, (O_1) ein, die Elemente weiter spezifizierendes, domänenspezifisches, aber testfallunspezifisches Typenmodell und (O_0) schließlich die Ebene spezifischer Konzepte in Form von Instanzen der Elemente der vorangegangenen Schicht. Diese letzte Ebene ähnelt funktional somit der *snapshot*-Beziehungen von Objekten zu Klassen, wie sie häufig bei der MOF-konformen Modellierung auf der Ebene M_0 genutzt wird, schafft aber darüber hinaus eine klare Trennung der betroffenen Elemente. Durch Integration der ontologischen Modellierungshierarchie in die linguistische ergibt sich das in Abbildung 49 dargestellte Bild geschachtelter Modellierungshierarchien.

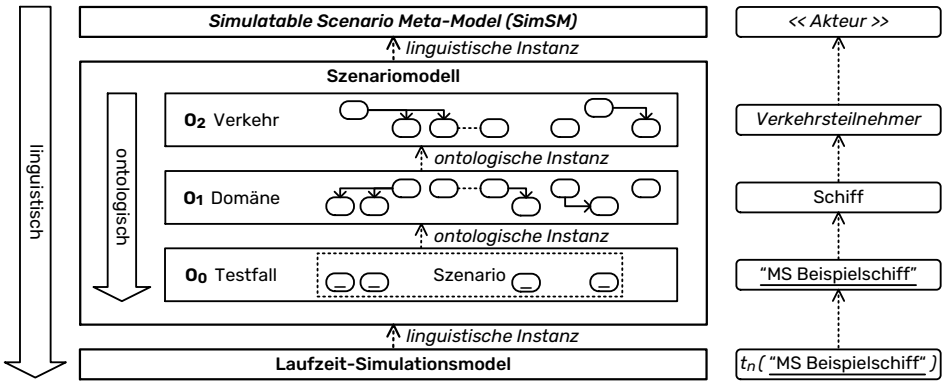


Abbildung 49: Ontologische Modellhierarchie eingebettet in die Ebene der linguistischen Modellhierarchie. Durch die konzeptorientierte Trennung der Modellinhalte auf die Schichten O_2, O_1 und O_0 wird die Wiederverwendung der tieferen Schichten ermöglicht. Auf der rechten Seite ist jeweils ein Beispiel-Element für jede Schicht abgebildet.

Auch innerhalb eines Domänenmodells sind zumeist hierarchische Vererbungsbeziehungen zu finden, die sich in Form einer eindeutigen Ontologie ordnen lassen (vgl. Abbildung 50). Um auch dieses Wiederverwendungspotenzial auszuschöpfen, ist die ontologische Ebene O_1 ein weiteres Mal unterteilt. Die hier mit D_n identifizierten Teilmodellpakete sind in ihrer Anzahl nicht begrenzt, da beliebig viele Spezialisierungs-Ebenen existieren können. Die Voraussetzung für einen sinnvollen Einsatz einer solchen Unterteilung ist aber, dass sich alle Elemente der Ontologie umfassend klar voneinander abgegrenzte Ebenen identifizieren lassen, die eine Detaillierung oder Spezifizierung der Abbildung der Wirklichkeit darstellen. Im Rahmen der Erarbeitung einiger Modelle hat sich für die Modellierung von Verkehr eine dreistufige Paketierung D_1 bis D_3 als sinnvoll und breit anwendbar hervorgetan: Verkehrsart (Wasser, Land …), Verkehrsträger (Straße, Schiene …) und Typ (Transport von Gütern, Radverkehr …). Die Integration dieser drei Pakete in die zuvor beschriebene Modellierungshierarchie ist in Abbildung 51 dargestellt.

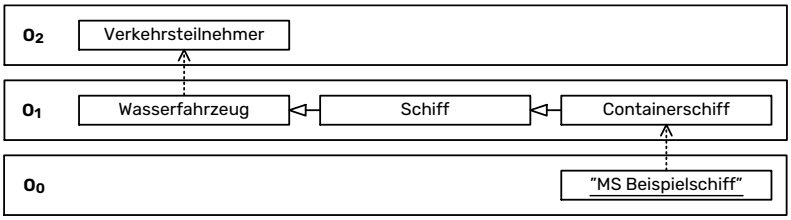


Abbildung 50: Mögliche inhaltliche Vererbungsstruktur innerhalb der Domänen-Ebene

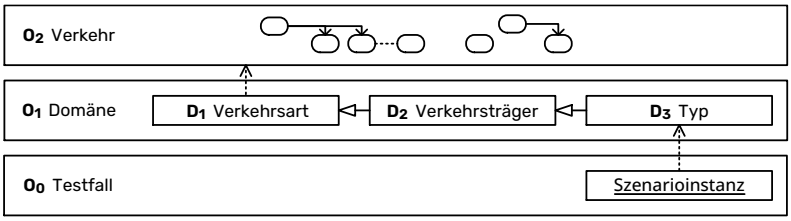


Abbildung 51: Mögliche Aufteilung der Elemente des Domänenmodells auf die drei Teilpakete D_1 bis D_3

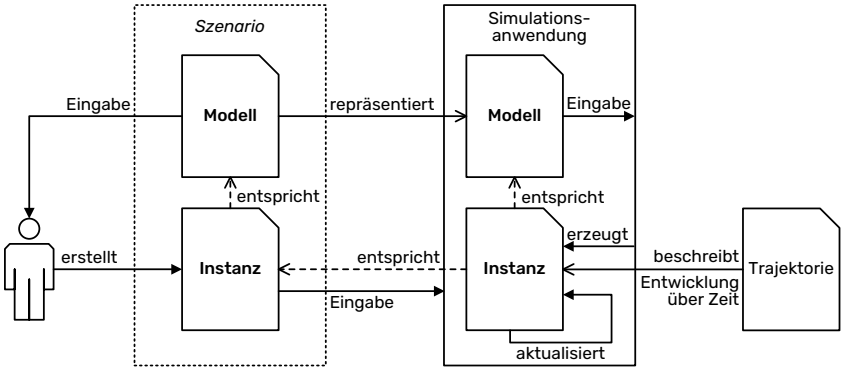


Abbildung 52: Referenz- und Nutzungsbeziehungen der Modelle eines szenariobasierten Simulationslaufs.

14.1.4 TRANSIENTES SIMULATIONSMODELL

In Abschnitt 12 wurde bereits festgestellt, dass die Forderung nach Flexibilität nicht alleine durch einen entsprechenden Modellierungsansatz zu erfüllen ist. Vielmehr muss die Simulationsanwendung in der Lage sein, Szenarioinstanzen ebenso flexibel zu simulieren. Die in den Abschnitten 7.4.2 und 7.4.3 beschriebene Repräsentationsbeziehung zwischen Szenario- und Simulationsmodell wird durch Abbildung 52 in die Systemumgebung szenariobasierter Simulationsläufe eingeordnet. Es ist zu erkennen, dass Anpassungen des Szenariomodells, wie sie zur Erfüllung des Teilziels 1 nötig sind, ebenfalls zu notwendigen Anpassungen des, der Simulationsanwendung inhärenten, Simulationsmodells führen.

Bei Betrachtung der zuvor identifizierten orthogonalen Beziehung der Modellierungshierarchien (vgl. Abbildung 39) fällt auf, dass einige funktional äquivalente Elemente in beiden Hierarchien existieren. Eine Linearisierung der Abbildungsbeziehungen durch Entfernen der gedoppelten Elemente aus der Modellierungshierarchie des Simulationsmodells ist somit möglich und bietet das Potenzial, die Probleme, die sich aus der Abhängigkeit der beiden Modelle ergeben, zu lösen. Da in diesem Zustand zwei Wirklichkeiten Teil der Gesamthierarchie wären, müssen die zuvor in separaten Hierarchien existierenden Elemente integriert und harmonisiert werden. Bei dieser Betrachtungsweise ist das Laufzeitmodell der Simulationsanwendung ebenfalls auf der *Modell*-Ebene zu verordnen und die Wirklichkeit dieser singulären Gesamthierarchie ist die Realität (vgl. Abbildung 53). Eine Besonderheit dabei ist, dass sowohl die Szenarioinstanz als auch das Laufzeitmodell durch eine Snapshot-Beziehung (vgl. Ende des vorigen Abschnitts 14.1.3) zur Menge der Modell-Elemente definiert sind.

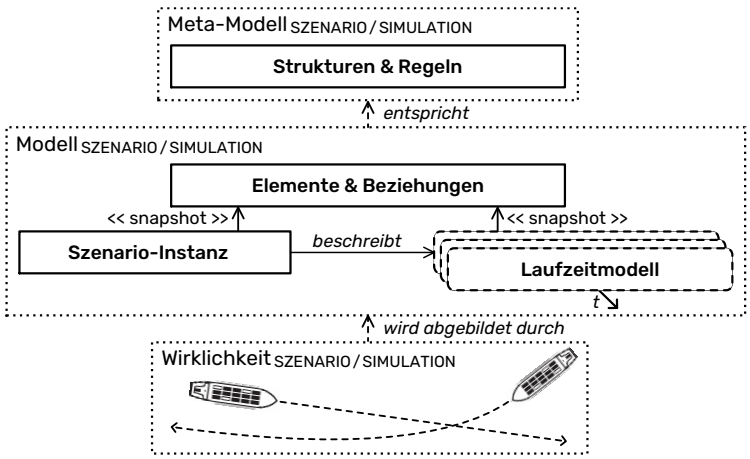


Abbildung 53: Zusammengeführte Szenario- und Simulationsmodelle. Das Laufzeitmodell wird mit Hilfe der Szenarioinstanz aus den Modellelementen erzeugt und existiert nur flüchtig (transientes Simulationsmodell).

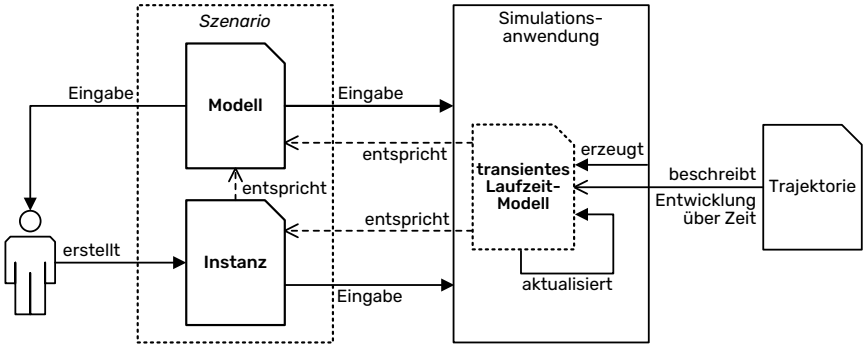


Abbildung 54: Transientes Simulationsmodell

Um die Flexibilität des Szenariomodells vollständig auszuschöpfen, ist das Laufzeitmodell nicht als dauerhaftes und persistentes Modell zu verstehen. Stattdessen wird es zu Beginn eines Simulationslaufs aus den vorgelagerten Modellelementen entsprechend der Szenarioinstanz erzeugt und existiert lediglich flüchtig während des Simulationslaufs (vgl. Abbildung 54). Die Simulationsanwendung selbst beinhaltet somit kein eigenes Modell und hat keine Kenntnisse über die Inhalte potenzieller Simulationsläufe. Stattdessen stellt sie lediglich eine Art Abspieler für Szenarien und dazugehörige Modelle dar, wodurch bei Anpassungen der Modelle und Szenarioinstanzen alle programmatischen und administrativen Arbeiten an der Simulationsanwendung selbst entfallen. Das Konzept, Szenarioinstanzen und Laufzeitmodelle als gleichwertige Abbildung der Realität zu sehen und Laufzeitmodelle lediglich flüchtig zu halten, wird im weiteren Verlauf als *transientes Simulationsmodell* bezeichnet und stellt zusammen mit der entsprechenden Realisierung (vgl. Abschnitt 14.3 und 15) ein zentrales Element zur Beantwortung der Forschungsfragen dieser Arbeit dar.

14.1.5 KOMPLEXITÄTSREDUKTION

Das gestaltungsorientierte Teilziel 3 fordert, Bezug nehmend auf die Teilforschungsfrage 3, dass die Einstiegshürde für potenzielle Nutzer der Simulationsanwendung im vorgesehenen Anwendungsumfeld reduziert werden muss. Die Höhe ergibt sich hauptsächlich aus dem Aufwand, der nötig ist, um Testszenarien zu erstellen und zu simulieren [Lou et al., 2022] (vgl. Abschnitt 5). Zusätzlich hat sich während der explorativen Phase dieser Arbeit gezeigt, dass sich eine unklare Rollenverteilung und eine fehlende Abgrenzung von Verantwortlichkeiten innerhalb des szenariobasierten simulativen Testprozesses negativ auf die Einsetzbarkeit auswirken. Die nachfolgenden Teilkonzepte adressieren daher sowohl die Kapselung von technischen Komplexitäten als auch die Ermöglichung der klaren Trennung der Verantwortlichkeiten der in Abschnitt 13.1 definierten Nutzerrollen.

14.1.5.1 MODULARISIERUNG DER MODELLE

Um die in Abschnitt 13.1 definierte Trennung der Verantwortlichkeiten vollständig zu ermöglichen, müssen die von den jeweiligen Aktivitäten betroffenen Modell- und Simulationsbestandteile als voneinander getrennte persistente Artefakte vorliegen. Die entsprechenden Inhalte der zuvor beschriebenen Modellierungshierarchie wurden zu diesem Zweck, Bezug nehmend auf die in Abschnitt 13.1 beschriebenen Rollen und Aktivitäten, wie in Abbildung 55 dargestellt, identifizierend zugeordnet.

Simulationsentwickler: Der Simulationsentwickler ist verantwortlich für die Bereitstellung der Simulationsanwendung. Diese hält nach Abschnitt 14.1.4 kein eigenes persistentes Modell vor und stellt lediglich einen „Abspieler“ für extrinsisch zugeführte Szenarien und Modelle dar. Somit hat der Simulationsentwickler keine Verantwortlichkeiten bezüglich des eigentlichen Modells. Da für die

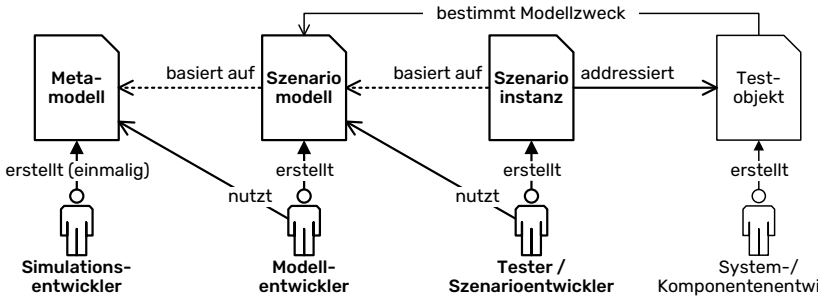


Abbildung 55: Verantwortlichkeiten in Bezug auf die Teilmodelle der Modellierungshierarchie

Interpretation von Modellen das Metamodell die grundlegenden Fähigkeiten der Simulationsanwendung abbilden muss, ist er verantwortlich für die Bereitstellung des *Metamodells*. Sowohl die Entwicklung der Simulationsanwendung als auch die Entwicklung des Metamodells sind einmalige Tätigkeiten.

Modellentwickler: Der Modellentwickler ist dafür verantwortlich, basierend auf dem Metamodell auf den jeweiligen Modellzweck ausgerichtete *Modelle* zu erstellen oder bestehende Modelle entsprechend auf den jeweiligen Modellzweck ausgerichtet anzupassen.

Tester: Der direkte Anwender der Simulationsanwendung ist der Tester. Er nutzt die vom Modellentwickler bereitgestellten Modelle, um durch Auswahl und Parametrierung der Modell-Elemente konkrete *Szenarioinstanzen* zu spezifizieren.

System-/Komponentenentwickler: Der System-/Komponentenentwickler ist verantwortlich für die Entwicklung und Umsetzung des simulativ zu testenden *Testobjekts*. Er hat zwar keine direkten Berührungspunkte mit den Modellen, das von ihm geschaffene Artefakt bestimmt jedoch den Modellzweck und infolgedessen den Umfang und die Granularität des Modells.

Entsprechend dieser Rollenverantwortlichkeiten sind die getrennt voneinander zu betrachtenden und persistent zu haltenden Modell-Module somit das Metamodell, das Szenariomodell und die Szenarioinstanz. Abbildung 56 zeigt eine entsprechende Projektion der in Abbildung 53 dargestellten Modell-Teile auf die identifizierte Modulstruktur.

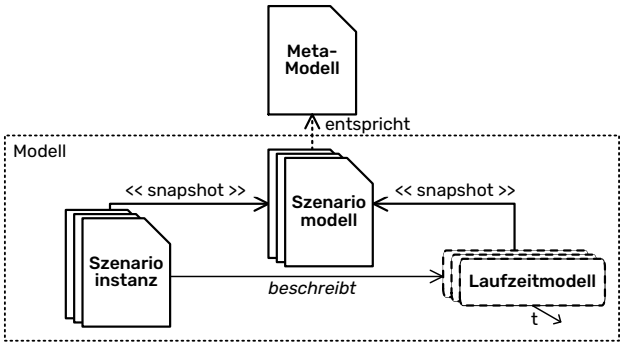


Abbildung 56: Modularisierung der Modellierungshierarchie

14.1.5.2 AGENTENGERÜST

Die Abschnitte 14.1.2 und 14.1.3 haben Konzepte zur Steigerung der Wiederverwendbarkeit von Teilmodellen durch die strukturierte Repräsentation von inhaltlichen und verhaltensbezogenen Aspekten vorgestellt. Zusätzlich muss auch eine struktur- und prozessorientierte Vereinheitlichung und Wiederverwendbarkeit geboten werden, um die angestrebte Komplexitätskapselung zu erreichen.

So muss der Modellentwickler Modelle weitestgehend losgelöst von den, der Ausführung dieser zugrunde liegenden, Simulationstechniken, -strukturen und -abläufen erstellen können. Indes bedarf es aber u. a. eines Mechanismus, um Ereignisse entsprechend ihrer kausalen Reihenfolge zu ordnen und die Synchronität der Zustände der simulierten Elemente zu gewährleisten [Bononi et al., 2005]. Um die Notwendigkeit des Kontakts des Modellentwicklers mit entsprechenden Mechaniken gemäß der definierten Rollen- und Verantwortungsverteilung zu verhindern, wird ein Agentengerüst vorgegeben, welches alle Mechanismen kapselt, die für die Interaktion des Agenten mit der Multi-Agenten-Simulation notwendig sind. Abbildung 57 zeigt den grundsätzlichen Aufbau dieses Agentengerüsts. Durch wenige einheitliche Schnittstellen werden die für die Umsetzung des Verhaltens notwendigen Funktionalitäten bereitgestellt: das Erhalten von Informationen über Aktualisierung in der Agentenumgebung (Eingabe) und das Beeinflussen der Agentenumgebung und der eigenen Repräsentation in dieser durch das Bereitstellen oder Aktualisieren von Daten (Ausgabe). Zusätzlich werden Mechanismen der Simulationssteuerung, wie die erwähnte Synchronisation, durch das Agentengerüst übernommen, sodass das Verhaltensmodell auf der obersten Ebene (vgl. Abschnitte 14.1.1 und 14.1.2) lediglich die Anforderung erfüllen muss, dass eine Routine vorhanden ist, welche vom Agentengerüst bei jedem Fortschreiten der Gesamtsimulation zur Berechnung des Agentenverhaltens angestoßen werden kann.

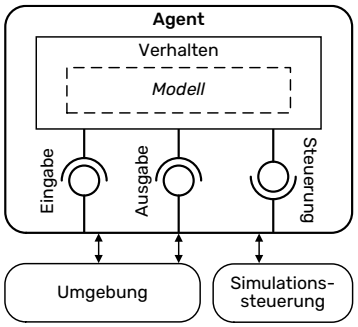


Abbildung 57: Agentengerüst zur Kapselung der Kommunikations- und Steuerungskomplexität

Trotz möglichst starker Reduzierung der Menge der ablaufspezifischen Details, mit denen sich der Modellentwickler auseinandersetzen muss, darf das Agentengerüst nicht zu einschränkend sein. Die vorgeschriebenen Strukturen und Schnittstellen müssen, ähnlich der in [Sloman & Logan, 1999; Sloman & Poli, 1996] skizzierten Idee, ausreichend generisch sein, damit verschiedene agenteninterne Modelle und Paradigmen entsprechend des Modellzwecks eingesetzt werden können. Es existiert eine Vielzahl von Ansätzen, unter anderem dazu, wie der interne Zustand eines Agenten und der Aktualisierungsprozess dessen gestaltet werden können, wie Sensoren und Aktoren programmatisch repräsentiert sein können und wie der Entscheidungsfindungsprozess ausgestaltet sein kann. Zwei Beispiele für eine solche konkrete Strukturierung des Agenteninneren sind *Beliefs, Desires and Intentions* (BDI) und *Cognition, Communication, Actuators, Sensors, Motivation* (COSY). Die ablaufunspezifisch ausgestalteten Schnittstellen (vgl. Abbildung 57) sind daher die einzigen durch das Agentengerüst vorgeschriebenen Strukturen, die es einzuhalten gilt (vgl. 14.3 und 15). Somit bleibt die konkrete Ausgestaltung interner Datenstrukturen und Abläufe dem Modellentwickler überlassen.

14.1.5.3 VORDEFINIERTE TEILMODELLE

Ein Ansatz, der von vielen Softwarewerkzeugen verfolgt wird, die für bestimmte klar definierte Bereiche entwickelt wurden, ist die Beschränkung der Inhalte des zugrundeliegenden Modells auf eben jenen Anwendungsbereich. Durch diese starke Einschränkung sind sehr benutzerfreundliche Konfigurationsumgebungen möglich. Diese Werkzeuge kommen mit Bibliotheken vordefinierter Modelle bestimmter Bereiche, wie dem Entwurf von Schaltplänen oder dem Modellieren von chemischen Reaktionen. Diese Bibliotheksinhalte können dann unter Beibehaltung der Topologie durch die Nutzer instanziiert und miteinander verbunden werden. [Ramos & Piera, 1999]

Um die Wiederverwendbarkeit und Nutzbarkeit weiter zu erhöhen, wurde der Kern dieser Idee aufgegriffen und zusätzlich das Konzept einer Domänenbibliothek eingeführt. Eine Domänenbibliothek ist hier eine teilparametrierte Teilmenge der Elemente der Domänenschicht. Sie ermöglicht es, vorparametrierte Elemente für die Szenariospezifikation in Form eines metaphorischen „Baukastens" anzubieten. So können etwa konkrete Schiffe verschiedener Klassen vordefiniert werden („Bausteine"), auf die der Tester bei der Szenariospezifikation zurückgreifen kann. Die Bibliothek ist, da es sich bereits um (abstrakte) Instanzen von Modellelementen handelt, auf der Testfall-Ebene eingeordnet und hat einen existenzbegründenden Bezug zum Modellzweck. Die Szenarioinstanz steht als Zielmenge in einer linkseindeutigen und rechtsvollständigen Relation zur Bibliothek, was durch einen durchgezogenen unidirektionalen Pfeil als *snapshot*-Beziehung dargestellt wird. Abbildung 58 zeigt die zuvor in Abschnitt 14.1.3 eingeführte Modellschicht der übergeordneten Hierarchie (vgl. Abbildung 51) entsprechend um die Szenariobibliothek erweitert.

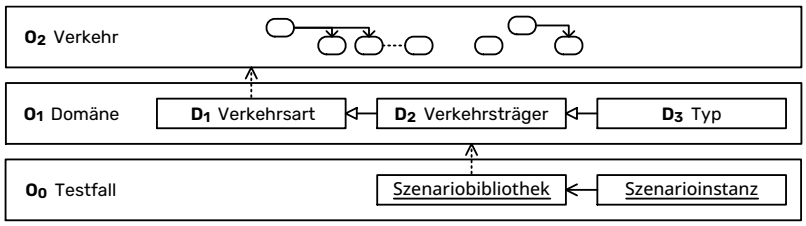


Abbildung 58: Szenariobibliothek als Basis für die Spezifikation der Szenarioinstanz

Die eingeführte Bibliothek als Zwischenschritt zwischen ontologischem Domänenmodell und der Szenarioinstanz ermöglicht es, basierend auf derselben D_3-Elementmenge verschiedene Ausprägungen für die Spezifikation von Szenarioinstanzen anzubieten. Sie bietet so u. a. das Potenzial, das Modell weiter auf einen bestimmten zulässigen *Betriebsbereich* (engl.: Operational Design Domain, ODD) (vgl. [Da Bruto Costa et al., 2025]) des Testobjekts einzugrenzen und damit die Nutzbarkeit und Wiederverwendbarkeit des Domänenmodells zu erhöhen (vgl. [Balci et al., 2017]).

14.1.6 TESTOBJEKT ALS TEIL EINER SZENARIOINSTANZ

Der dritte große Themenbereich dieser Arbeit, eröffnet durch die Teilforschungsfrage 3 und adressiert durch die gestaltungsorientierten Teilziele 4 und 5 (vgl. Abbildung 41), ist die Ermöglichung der aktiven Teilnahme des externen Testobjekts am Simulationslauf. Auch hier ist das zielführende Konzept zweitgeteilt erarbeitet worden: Nachfolgend wird die inhaltliche Integration in das Szenariomodell beschrieben und anschließend in 14.1.7 erweiternd die architekturelle Ermöglichung des nötigen Datenaustausches zwischen dem realen Testobjekt und den Elementen der simulierten Welt.

Um eine einheitliche Integration des Testobjekts in Szenarioinstanzen zu ermöglichen, wurde das in Abschnitt 14.1.2 vorgestellte Metamodell zur Modellierung simulierbarer Szenarien (vgl. Abbildung 47) entsprechend erweitert. Der von der Erweiterung betroffene Bereich des Metamodells ist in gekürzter Form in Abbildung 59 dargestellt. Aufgrund dessen, dass der gesamte Entwicklungsprozess unterstützt werden soll, muss es möglich sein, Testobjekte aller Hierarchieebenen eines automatisierten Fahrzeugsystems (vgl. Abschnitt 6.3, insb. Abbildung 14) zu integrieren: (Fahrzeug-)Systeme, (Teil-)Systeme, Komponenten, Subkomponenten und Einheiten.

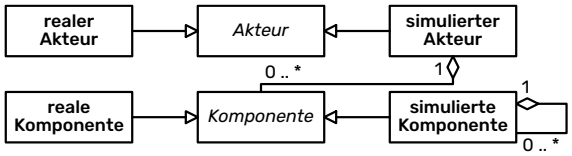


Abbildung 59: Erweiterung der möglichen Szenarioinhalte um eine Repräsentation der möglichen Testobjekte

Die Metamodell Elemente *Akteur* und *Komponente* haben zur Repräsentation der realen Testobjekte jeweils mittels Vererbung eine zusätzliche Variante erhalten: *realer Akteur* und *reale Komponente*. Um eine einheitliche Struktur zu wahren und die Zusammensetzungsverhältnisse korrekt abbilden zu können, sind *Akteur* und *Komponente* abstrakte Konzepte und die zu simulierenden Pendants, *simulierter Akteur* und *simulierte Komponente,* werden analog mittels Vererbung als Variante gehandhabt. Ein *Akteur* kann somit entweder ein *realer Akteur* oder ein *simulierter Akteur* sein. Ein *simulierter Akteur* kann beliebig viele *Komponenten* zugeordnet haben. Handelt es sich dabei um eine *simulierte Komponente*, können dieser wiederum ebenfalls eine beliebige Anzahl untergeordneter *simulierter Komponenten* zugeordnet sein.

Teilsysteme, Komponenten sowie Hardware-/Softwareeinheiten können aus Modellierungssicht vereinheitlicht werden, d. h., die Wirkbeziehung zwischen Eingaben und Ausgaben ist bei allen Elementen, die sich nicht auf der obersten Ebene der Wirkhierarchie befinden, identisch. Daher kann das Element *reale Komponente* zur Abbildung aller untergeordneten Ausprägungen von Testobjekten genutzt werden. Die einzelnen Abbildungsbeziehungen aller in Abbildung 14 dargestellten relevanten realen Elemente zu den beiden eingeführten Metamodell-Elementen zeigt Tabelle 13.

Ein *simulierter Akteur* kann somit eine im Rahmen einer Szenarioinstanz abgebildete interne Wirkstruktur besitzen, aber ein *realer Akteur* und eine *reale Komponente* nicht. Selbstverständlich können auch reale Systeme sowie reale Komponenten untergeordnete Komponenten(-hierarchien) besitzen. Im vorliegenden Fall ist jedoch nur die oberste Ebene zu berücksichtigen, da externe Testobjekte als Blackbox zu betrachten sind (vgl. Abschnitte 3 und 6.3) und die internen Details somit für die Integration in ein zu simulierendes Szenario nicht relevant sind.

Tabelle 13: Abbildung der Elemente der Wirkbeziehung realer Fahrzeuge durch die Elemente des Metamodells

Element der realen Struktur	Abgebildet durch
(Fahrzeug-)System	realer Akteur
(Teil-)System	reale Komponente
Komponente	reale Komponente
Hardware-Einheit / Software-Einheit	reale Komponente

14.1.7 DAS TESTOBJEKT ALS TEILNEHMER AM SIMULATIONSLAUF

Damit Systeme und Komponenten simulativ getestet werden können, reicht es natürlich nicht aus, die virtuelle Umwelt des Testobjekts in Form von Szenarioinstanzen konfigurieren und simulieren zu können. Die Grundvoraussetzung ist zunächst einmal die Teilnahme des Testobjekts am entsprechenden Simulationslauf. Das heißt, dass das Testobjekt Eingangssignale aus seiner simulierten Umwelt erhält und die auf Basis dieser erzeugten Ausgangssignale auch in die simulierte Umwelt zurückspielen kann (vgl. Abschnitt 6.3). Die zu bewältigenden Herausforderungen, um eine entsprechende Interoperabilität zu schaffen, lassen sich dabei auf zwei Bereiche aufteilen: *syntaktische Interoperabilität* und *semantische Interoperabilität* [Gütlein, 2023; Evora Gomez et al., 2016]. Erste behandelt die technischen Aspekte der Kommunikation zwischen den zu verbindenden Komponenten, wie das Etablieren eines Kommunikationskanals, das Packaging der Daten, das Versenden der Daten und das Entgegennehmen der Daten. Die Möglichkeit des Austauschs von Daten alleine ermöglicht aber noch keine sinnvolle Interoperabilität, weshalb auch dafür Sorge zu tragen ist, dass die kommunizierenden Komponenten bezüglich der übertragenen Daten auch dieselbe Sprache sprechen oder zumindest die des jeweils anderen eindeutig und korrekt interpretieren können. Um das gestaltungsorientierte Teilziel 5 zu erfüllen, sind somit Konzepte zur Sicherstellung von sowohl syntaktischer als auch semantischer Interoperabilität zwischen der Simulationsanwendung bzw. den Modellelementen zur Laufzeit und dem externen Testobjekt nötig.

Syntaktische Interoperabilität

Da das Testobjekt in den meisten Fällen nicht auf demselben Computersystem ausgeführt wird wie die Simulationsanwendung oder Teil dessen ist, muss es die Möglichkeit für einen Datenaustausch über die Grenzen des simulierenden Computersystems (Simulationshost) hinaus geben. Eine entsprechend flexibel einsetzbare Möglichkeit ist die Kommunikation über Netzwerkverbindungen anhand standardisierter Netzwerkprotokolle (vgl. Abbildung 60). Idealerweise ist das Testobjekt Teil desselben lokalen Netzwerks (engl.: Local Area Network, LAN) wie der Simulationshost. Vor allem im Fall von HiL- und ViL-Tests kann dies allerdings häufig alleine aufgrund der physischen Größe nicht garantiert werden. Zusätzlich kann es sein, dass das Testobjekt aus Sicherheitsgründen in einem anderen Netzwerk lokalisiert ist als der Simulationshost. Auch sehr frühe Entwicklungszustände oder das Bestreben nach Schutz von geistigem Eigentum können Gründe dafür sein, dass die Bereitstellung des Testobjekts im lokalen Netzwerk des Simulationshosts keine vertretbare Option darstellt. Somit muss neben der Kommunikation innerhalb eines lokalen Netzwerks auch die Möglichkeit zur Netzwerkkommunikation über die Grenzen unterschiedlicher lokaler Netzwerke sichergestellt sein.

Abbildung 60: Netzwerkkommunikation zwischen der Simulationsanwendung und dem externen Testobjekt.

Aus mehreren Gründen bietet sich der Aufbau der Simulationsanwendung als verteilte Simulation an:

Flexible Verteilung: Verteilte Simulationen befassen sich mit der Ausführung von Computersimulationen über mehrere Computersysteme hinweg. Dabei liegt der Fokus primär auf der Abdeckung größerer geografischer Bereiche, als es beim reinen Parallel Computing üblicherweise der Fall ist. Eine ver-

teilte Simulation kann von einer Menge von Computersystemen ausgeführt werden, die über ein lokales Netzwerk miteinander verbunden sind, oder auch von solchen, die weltweit verteilt sind und über das Internet miteinander kommunizieren. Dabei ist das vorrangige Ziel, dass die Simulationsinhalte der *Simulationseinheiten* (engl.: Simulation Unit, SU), die auf voneinander unabhängigen Computersystemen laufen, zu einer einzelnen Simulationsumgebung integriert werden. [Fujimoto, 2015]

Ein softwarebasiertes Testobjekt kann ebenfalls als eine Simulationseinheit angesehen werden, solange es Ausgangssignale auf Basis von Eingangssignalen erzeugt und damit ein Verhalten beschreibt. Somit deckt sich der vorliegende Fall der Anbindung eines geografisch möglicherweise weit entfernten softwarebasierten Testobjekts an die Simulationsanwendung mit der Definition und den Zielen einer verteilten Simulation. Da verteilte Simulationen, vorwiegend der Spezialfall der *Kooperativen Simulation* (engl.: Cooperative Simulation, Co-Simulation) (vgl. [Geimer et al., 2006]), die einzelnen Simulationseinheiten als Blackbox betrachten [Gomes et al., 2019] und die Integration über die Eingangs- und Ausgangssignale erfolgt, kann auch die Sicherheit geistigen Eigentums gewahrt werden.

Vorhandene Standards: Es existieren einige Standards und Regelwerke zur Realisierung verteilter und kooperativer Simulationen. Diese adressieren die notwendigen Aspekte syntaktischer Interoperabilität, wie einen effizienten Datenaustausch und die Sicherstellung von Synchronisation, weitestgehend vollständig. Einige dieser Standards sind etabliert und weit verbreitet und können daher eine Möglichkeit bieten, die grundsätzliche technische Funktionalität sicherzustellen. Auf dieser Basis sollte aufgebaut werden, statt eine fehleranfällige vollständige Neuentwicklung anzustreben.

Agentenmetapher: Die für die Modellierung gewählte Agentenmetapher deckt viele der Anforderungen einer verteilten Ausführung [Abar et al., 2017] ab und lässt sich somit mit lediglich geringem zusätzlichem Aufwand durch eine verteilte Simulationsanwendung abbilden: Jeder Agent interagiert selbstständig mit seiner Umwelt über Eingangs- und Ausgangsgrößen, besitzt einen eigenen selbstverwalteten internen Zustand und seine innere Struktur kann beliebiger Ausprägung sein. Schlussfolgernd bietet es sich daher an, auf dieser Ebene ebenfalls nicht nur im Rahmen der Modellierung zu trennen, sondern auch infrastrukturell.

Lastverteilung: Ein Hauptmotivator für die Entwicklung verteilter Simulation war und ist die Lastverteilung durch Parallelisierung [Fujimoto, 2003]. Dadurch ergeben sich zwei Vorteile: (1) die schnellere Ausführung komplexer Simulationen durch die Verteilung der Berechnungen auf mehrere Computersysteme, (2) die Ermöglichung der Durchführung größerer Simulationen, als ein einzelnes Computersystem erlauben würde, durch die Nutzung von mehr Ressourcen durch Parallelisierung auf mehrere Computersysteme.
Zur Ausführung einfacher Verkehrsszenarien genügt oftmals ein einzelner (Arbeitsplatz-)Rechner. Für die simulative Ausführung sehr umfangreicher Szenarien mit vielen dynamischen Elementen, deren Wirkstruktur feingranular modelliert ist und deren Verhalten komplexe, realitätsnahe Berechnungen umfasst, kann die Möglichkeit zur verteilten Ausführung aber einen Mehrwert bieten.

Bezüglich der möglichen Integration eines Testobjekts in eine verteilte Simulationsanwendung wurden zwei Möglichkeiten als relevant identifiziert (vgl. Abbildung 61 und Abbildung 62): (1) Das Testobjekt repräsentiert einen Akteur, befindet sich somit auf der höchsten Ebene der hierarchischen Wirkstruktur eines modellierten Fahrsystems, erfüllt aber nicht die technischen Voraussetzungen, um direkt als Simulationsteilnehmer in den verteilten Simulationslauf eingebunden zu werden; (2) Das Testobjekt

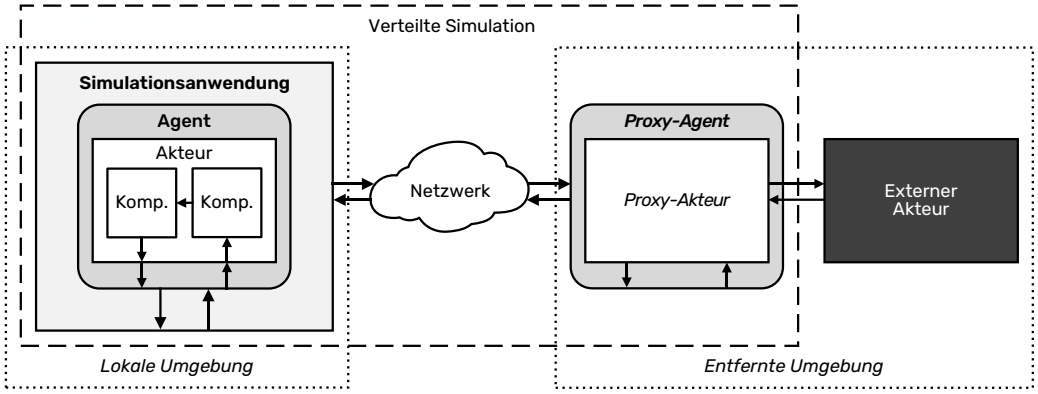


Abbildung 61: Anbindung eines externen Testobjektes in der Rolle eines Akteurs

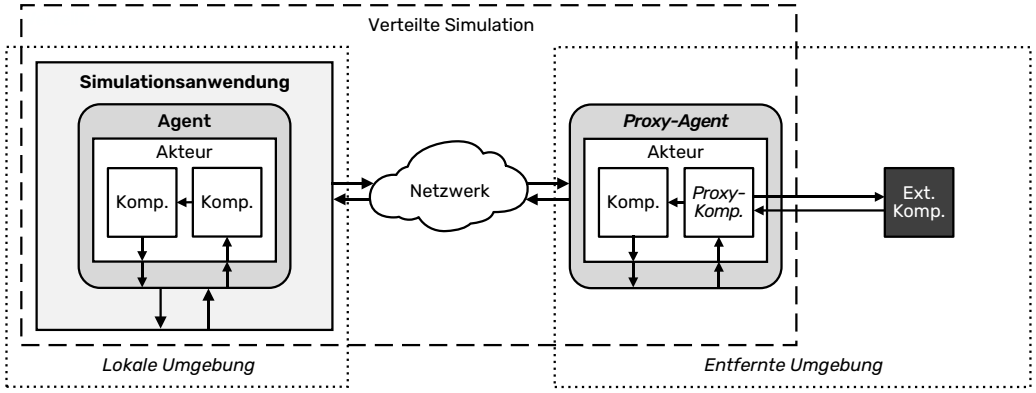


Abbildung 62: Anbindung eines externen Testobjektes in der Rolle einer Komponente

repräsentiert eine Komponente eines Akteurs, befindet sich also auf einer tieferen Ebene der hierarchischen Wirkstruktur eines modellierten Fahrsystems.

Um die zwei möglichen Erscheinungsformen von Testobjekten zu unterstützen, wurden entsprechend zwei architekturelle Lösungsansätze entworfen, welche die Simulationsanwendung unterstützen muss: (1) Das Testobjekt weist keine direkte Kompatibilität mit der verteilten Simulationsanwendung auf. Somit wird ein vermittelndes Element benötigt. Eine der Teil-Simulationen nimmt dafür die Rolle eines Stellvertreters (engl.: proxy) ein und repräsentiert einen *Proxy-Akteur* ohne eigenes dynamisches Verhalten. Dieser leitet Eingangssignale an das Testobjekt weiter und nimmt Ausgangssignale des Testobjekts entgegen. Der *Proxy-Agent* kann durch die Verteilung der Simulationsanwendung auf einem Computersystem im lokalen Netzwerk des Testobjekts verortet sein und wie gewohnt mit der verteilten Simulationsanwendung kommunizieren. Gleichzeitig erlaubt der *Proxy-Akteur* eine Kommunikation mit dem externen Testobjekt anhand von Protokollen, die für die direkte Integration in die Gesamtsimulation nicht geeignet sind oder nicht unterstützt werden. Entsprechende Anpassungen der verteilten Simulation oder des Testobjekts entfallen somit. (vgl. Abbildung 61); (2) eine Komponente befindet sich nicht auf der obersten Ebene der Wirkstruktur eines Fahrsystems und stellt in der definierten Simulationsumgebung keinen eigenständigen Agenten dar. Die Möglichkeit einer eigenständigen Teilnahme am verteilten Simulationslauf entfällt daher. Stattdessen steuert der *Proxy-Agent* analog zu Modell-Agenten einen durch die Szenarioinstanz beschriebenen modellierten Akteur mitsamt einer internen modellierten Wirkstruktur. Dabei wird an der dem Testobjekt entsprechenden Stelle der Wirkstruktur eine *Proxy-Komponente* eingesetzt, welche analog zum *Proxy-Akteur* den Datenaustausch mit dem Testobjekt übernimmt (vgl. Abbildung 62).

Semantische Interoperabilität

Standards und Frameworks für verteilte und kooperative Simulationen fokussieren sich in der Regel auf das Kombinieren mehrerer Simulationsanwendungen im Sinne syntaktischer Interoperabilität. Was sie zumeist nicht bieten, ist Unterstützung bei der Komposition der zugrunde liegenden Modelle. Um dafür zu sorgen, dass die Interpretationen der ausgetauschten Informationen auch übereinstimmen, ist zusätzlicher manueller Aufwand nötig. [Fujimoto, 2016]

In beiden identifizierten Fällen muss also Sorge dafür getragen werden, dass das externe Testobjekt und die Simulationsanwendungen die jeweils vom anderen empfangenen Daten interpretieren und nutzen können. Eine naheliegende mögliche Lösung dafür ist die Anpassung des Modells entsprechend des vom Testobjekt verwendeten und erwarteten Datenmodells. Eine 1:1-Beziehung von Modell und Testobjekt widerspricht jedoch deutlich der Zielsetzung einer hohen Wiederverwendbarkeit der Modelle. Somit ist die Anpassung des Modells hier keine valide Lösungsvariante. Stattdessen werden die vorhandenen Proxy-Elemente weiter in die Verantwortung gezogen und um eine bidirektionale Übersetzungsfunktionalität erweitert.

Ein einfaches Beispiel für ein semantisches Interoperabilitätsproblem wird in [Evora Gomez et al., 2016] beschrieben: An einer kooperativen Simulation nehmen zwei Simulationseinheiten teil. Eine repräsentiert ein *Gebäude* und die andere die *Umgebung*, in der sich das Gebäude befindet. Das Modell der *Gebäude*-Simulationseinheit nutzt für Temperaturen die Einheit Fahrenheit (F) und das Modell der *Umgebung* die Einheit Celsius (C). Die beiden Modelle könnten zwar technisch kooperieren, es würde aber zu falschen Berechnungen kommen, da ein von der *Umgebung* als Ausgangsgröße zur Verfügung gestellter Wert von 10 vom *Gebäude* fälschlicherweise als 10 °F (ca. –12 °C) interpretiert werden würde, statt dem korrekten Wert von 50 °F (10 °C). Evora Gomez et al. schlagen zur Lösung die Definition von Zusammenhängen von Ein- und Ausgangsgrößen sowie der bidirektionalen Transformation innerhalb der Konfigurationsdatei der kooperativen Simulation vor. Daran angelehnt wird die Abbildung des Testobjekts innerhalb einer Szenarioinstanz (vgl. Abschnitt 14.1.6) erweitert um das Element *dictionary* (dt.: Wörterbuch). Innerhalb dieses Elements kann der Szenarioentwickler die Eingangs- und Ausgangssignale und die Zusammenhänge zwischen diesen definieren, die für die bidirektionale Kommunikation zwischen Simulationsanwendung und Testobjekt relevant sind. Das entsprechende Proxy-Element übernimmt während des Simulationslaufs die Transformation sowohl der Ausgangs- als auch der Eingangssignale. Somit ist bei geringem zusätzlichen Aufwand sichergestellt, dass weder das Modell noch das Testobjekt angepasst werden müssen. Die genaue Ausgestaltung des *Dictionary* und des bidirektionalen Transformationsprozesses ist in Abschnitt 14.3.4 beschrieben.

14.1.8 ÜBERBLICK

Durch Zusammenführung der beschriebenen Teilkonzepte, welche jeweils mindestens eins der fünf gestaltungsorientierten Teilziele direkt adressieren, entsteht ein übergreifendes Gesamtkonzept zur Erfüllung des übergeordneten gestaltungsorientierten Ziels. Eine entsprechende Darstellung der Systemumgebung szenariobasierten simulativen Testens externer Testobjekte unter Einbeziehung der identifizierten Nutzerrollen sowie Integration der Konzepte zur hierarchischen Modellierung, Verteilung der Simulation und Anbindung des Testobjekts ist in Abbildung 63 zu sehen. Da die Konzepte bis zu diesem Punkt eher allgemeingültig und undetailliert beschrieben wurden, um so einen besseren Überblick über die Zusammenhänge innerhalb der entstandenen Systemumgebung zu gewähren, wer-

den im weiteren Verlauf in Abschnitt 14.3 ausgewählte Teilbereiche der Konzepte als Zwischenschritt auf dem Weg zur Beschreibung der experimentellen Umsetzung in Abschnitt 15 in Hinblick auf inhaltliche Detailfragen konkreter ausgearbeitet.

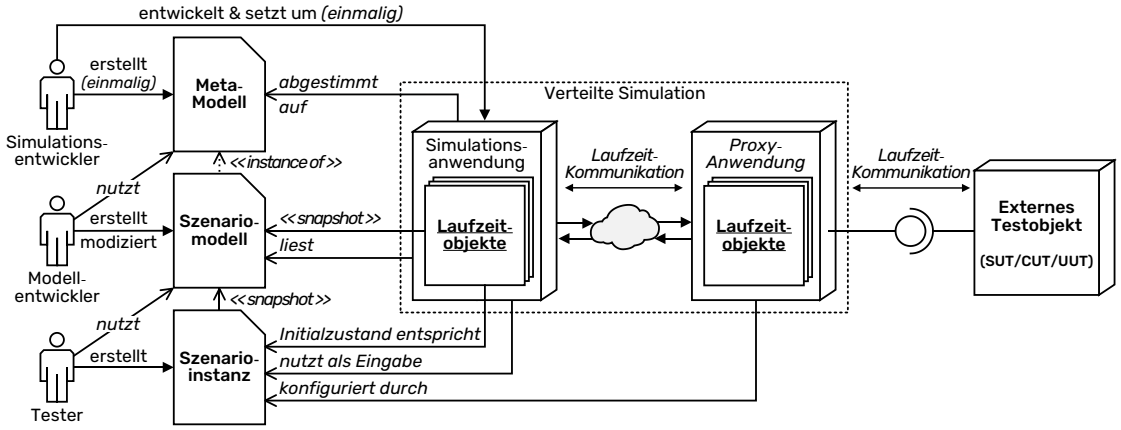


Abbildung 63: Überblick über die entworfene Systemumgebung szenariobasierter Simulationsläufe zu Testzwecken softwarebasierter Systeme oder Komponenten dieser.

14.2 VERWANDTE LÖSUNGSANSÄTZE

Bevor die konkrete inhaltliche Ausgestaltung der vorgestellten übergreifenden Konzepte vorgestellt wird, soll an dieser Stelle der Blick zunächst auf in der wissenschaftlichen Literatur existierende Lösungsansätze gerichtet werden, die ähnliche Teilherausforderungen adressieren wie die zentralen vorgestellten Konzeptteile. Dies dient der eindeutigen Abgrenzung der hier präsentierten Lösungsansätze von bereits existierenden, soll den Arbeiten aber gleichzeitig ebenfalls die verdiente Aufmerksamkeit zukommen lassen, da sowohl einige der Ideen und Konzepte als auch der Probleme und Lücken mindestens indirekt inspirierend auf die Ergebnisse dieser Arbeit eingewirkt haben.

14.2.1 HIERARCHISCHE SZENARIOMODELLIERUNG

Im Automobilbereich gibt es bereits seit einigen Jahren Bemühungen, Testszenarien durch eine Ordnung der möglichen Inhalte in teils hierarchischen Strukturen einheitlicher und modularer zu machen sowie systematische Testfallableitung zu ermöglichen. Vorgestellt in [Schuldt et al., 2013] und anschließend aufgegriffen und verfeinert in [Schuldt, 2017] wurde die erste Iteration eines Ebenen-Modells zur Beschreibung logischer und konkreter Fahrszenarien. Die vorgestellten Ebenen waren *Basisstreckennetzwerk*, *situationsspezifische Anpassungen am Basisstreckennetzwerk*, *Beschreibung und Regelung der Akteure* und *Umweltbedingungen*. Dieses Modell wurde in [Bagschik et al., 2018] und [Bock et al., 2018] um jeweils eine Ebene erweitert und in [Scholtes et al., 2021] weiter verfeinert. Von dem auf Straßenverkehr ausgerichteten Modell wurden zudem auch weitere Varianten abgeleitet, wie in [Wang et al., 2022] für Offroad-Verkehrs-Szenarien. Ähnliche Arbeiten in Bezug auf maritime Verkehrsszenarien konnten nicht gefunden werden.

Die Zielsetzung der genannten Modelle überschneidet sich bei genauerer Betrachtung nur minimal mit der des hier erarbeiteten Konzepts. Die genannten Modelle sind darauf ausgelegt, logische und/oder konkrete Szenarien zu erstellen. Somit haben sie nicht den Anspruch, als direkte Grundlage eines Simulationslaufs zu dienen. Dieser Umstand zeigt sich auch in der stark konzeptionell ausgerichteten Aufteilung der möglichen Inhalte auf die Ebenen, die sich stark an einer linguistischen Beschrei-

bung der realen Welt orientiert, statt an einer klaren hierarchischen Ordnung der Inhalte. So sind u. a. physische Objekte über alle Ebenen verteilt und ortsfeste Fahrzeuge der Ebene dynamischer Objekte zugeordnet. Im Gegensatz zu diesem schichtweisen Befüllen eines Szenarios mit Inhalten erfüllt die Modellhierarchie in dieser Arbeit den Zweck einer schrittweisen Spezifizierung der Modelle, basierend auf einem generischen Metamodell, mit dem Ziel, eine hohe Wiederverwendbarkeit zu gewährleisten. Die genannten Ebenen-Modelle sind inhaltlich zudem vollständig auf automobilen Verkehr ausgerichtet, welcher hier nicht im Fokus steht.

14.2.2 VERTEILTE SIMULATION

Die *e-Maritime Integrated Reference Platform (eMIR)*[64] umfasst sowohl ein physisches Testfeld als auch einen virtuellen Teil unter dem Namen *HAGGIS* (früher ein Akronym für *Hybrid Architecture for Granular, Generic and Interoperable Simulations*). Die Plattform soll das schnelle Testen neuer e-Navigationstechnologien in einer Simulationsumgebung ermöglichen und bietet dafür eine Reihe von Modulen wie die *Maritime Traffic Simulation (MTS)*, eine Sensorsimulation und eine einfache Umgebungssimulation [Hahn & Noack, 2016]. Die Verwendung einer Implementierung des *High Level Architecture* (HLA)-Standards als Kommunikations-Middleware ermöglicht die Integration externer Teilsimulationen, die ebenfalls HLA-konform umgesetzt sein müssen. Die MTS kann so konfiguriert werden, dass sie im Rahmen eines Simulationslaufs mit einem externen Testobjekt kommuniziert. Dies ermöglicht zwar theoretisch die Integration von Modellen (MiL), Software (HiL) und Hardware (HiL) in den Simulationslauf, jedoch ist die Verteilung der Simulation nicht stringent für die gesamte Plattform umgesetzt. Die Umsetzung der gesamten Verkehrssimulation in Form eines Simulationsteilnehmers erschwert die intuitive Integration eines externen Testobjekts als Akteur oder Komponente, da dieses so eine grundlegend andere Behandlung als die Modellkomponenten erfordert (vgl. [Schweigert et al., 2014]). Die tiefe und verzahnte Integration des zentralen *World-Models* in die meisten Module und funktionalen Bestandteile sowie die vielfältigen Abhängigkeiten machen die Plattform zu einer ungeeigneten Grundlage für die Erreichung der gesetzten Ziele. Stattdessen wird der virtuelle Teil von *eMIR* in Abschnitt 16.1.1 *Untersuchung der Eignung des Prototyps* als externes Testobjekt eine Rolle spielen.

Sadjina et al. [2019] skizzieren ein *Virtual-Prototyping-Framework* (VPF) für maritime Systeme und Operationen mit dem Ziel, die Entwicklung wiederverwendbarer Komponenten- und Teilsystemmodelle zu unterstützen und diese zu *Full-System*-Simulationen zu kombinieren. Als Hauptanwendungsfälle werden Prototyping, Verifikation, Training und Performance-Studien genannt. Das Ergebnis der Arbeit sind Leitfäden zur Modellkopplung, eine Reihe von Interfaces, eine eigene Simulationssoftware sowie einige Beispielmodelle und Demonstratoren. Die Motivation für die Konzipierung als verteilte Simulation weist Gemeinsamkeiten mit derjenigen dieser Arbeit auf: Blackbox-Nutzung (Schutz intellektuellen Eigentums) und Performance- und Nutzbarkeits-Aspekte. Der Fokus auf die Simulation einzelner umfangreicher Systeme durch Kombination der Systemkomponenten und die entsprechend auf diese *Full-System*-Simulation ausgerichtete Kopplungsarchitektur sind hingegen nicht geeignet, um einen agentenbasierten Ansatz umzusetzen, wie er hier angedacht ist.

[64] DLR Institute of Systems Engineering for Future Mobility, *eMIR – e-Maritime Integrated Reference Platform:* `https://www.dlr.de/en/se/research-transfer/research-infrastructure/emir` [Letzter Zugriff: 17.02.26]

Die Weiterentwicklung der zugrunde liegenden Simulationssoftware *Coral*[64] wurde zudem eingestellt. Als offizieller Nachfolger werden die Bibliotheken *libcosim* und *cosim* der *Open Simulation Platform* (OSP) genannt, welche im weiteren Verlauf in Abschnitt 14.3.1 noch weitere Erwähnung findet. *Libcosim* bietet derzeit zwar offiziell die Möglichkeit zur verteilten Ausführung an, diese ist aber über eine zusätzliche Schicht (*proxyfmu*) realisiert. Das bedeutet, dass die Distribution durch ein zusätzliches Vermittlungskonstrukt erfolgt und nicht als grundlegendes, natives funktionales Konzept im OSP-Ökosystem verankert ist. Daher wird *libcosim* an dieser Stelle nicht weiter betrachtet.

14.2.3 MODELLTRANSFORMATION

Ein Ansatz, der in den jüngst vergangenen Jahrzehnten zunehmend Beachtung gefunden hat, ist die modellbasierte Generierung verteilter Simulationsanwendungen. Da diese Arbeit mit dem Konzept des transienten Simulationsmodells eine ähnliche Richtung eingeschlagen hat, wird im Folgenden ein prominenter Ansatz in diesem Bereich beispielhaft abgrenzend vorgestellt.

Im Jahr 2012 schlugen Bocciarelli et al. eine modellgetriebene Methode für den Aufbau verteilter Simulationsanwendungen vor [Bocciarelli et al., 2012]. Die Arbeit konzentrierte sich zu diesem Zeitpunkt auf die Umwandlung von Geschäftsprozessmodellen in verteilte Simulationsläufe, um Analysen und Tests zu ermöglichen. Im selben Jahr wurde dieses Grundkonzept dahingehend erweitert, allgemeine Systemmodelle auf der Grundlage der Systems Modeling Language (SysML)[65] zu simulieren [Bocciarelli et al., 2012]. Die Idee besteht darin, ein SysML-Modell der geplanten kooperativen Simulation zu erstellen, das aus mehreren einzelnen Teilmodellen besteht. Diese werden dann mit technologiespezifischen Details (hier die *High Level Architecture (HLA)*, vgl. Abschnitt 14.3.1) unter Verwendung entsprechender UML-Profile angereichert und später durch einen mehrstufigen, teilautomatisierten Modelltransformationsprozess in eine abstrakte Implementierung überführt. Die resultierenden generierten Implementierungen sind lediglich Gerüste, die anschließend händisch mit Logik gefüllt werden müssen, um einen ausführbaren Simulationsverbund zu erhalten. Die manuelle Implementierung von HLA-spezifischen Abläufen wird so weitgehend vermieden, was den Entwicklungsprozess von HLA-basierten verteilten Simulationen weniger zeitaufwendig macht.

Dieser Ansatz wurde später weiter ausgebaut durch Hinzufügen von Cloud-basierten Funktionalitäten zur Bereitstellung [Bocciarelli et al., 2013], Harmonisierung mit dem *Model-Driven Architecture* (MDA)[66]-Standard [Bocciarelli et al., 2016] und Integration einer teilautomatisierten Generierung HLA-spezifischer Datenmodelle, um den Prozess der Erstellung einer vollwertigen ausführbaren verteilten kooperativen Simulation weiter zu vereinfachen [Bocciarelli et al., 2019]. Die aktuellste Iteration trägt den Namen *Model-Driven Distributed Simulation Development Process* (MoDSEEP) [Bocciarelli et al., 2020] und wurde an den standardisierten *Distributed Simulation Engineering and Execution Process* (DSEEP) [IEEE, 1730:2022] angepasst, der aus den Standardisierungsaktivitäten der HLA hervorgegangen ist und einen einheitlichen und rigorosen Prozess für die Entwicklung und Ausführung verteilter Simulationen bietet. Die Anwendbarkeit dieses modellgesteuerten Ansatzes wurde später anhand von Anwendungsfällen aus der Praxis und konkreten Werkzeugketten demonstriert [Kay et al., 2021; D'Ambrogio et al., 2020].

[65] Object Management Group (OMG), *Systems Modeling Language*: `https://sysml.org/sysml-specs/` [letzter Zugriff: 17.02.26]

[66] Model Driven Architecture (MDA), Object Management Group (OMG): `https://www.omg.org/cgi-bin/doc?ormsc/14-06-01` [letzter Zugriff: 17.02.26]

Obwohl der beschriebene Ansatz ähnliche übergeordnete Ziele wie die hier vorgestellten (Teil-)Konzepte hat, über viele Jahre verbessert und erweitert wurde und praktisch anwendbar zu sein scheint, unterscheidet sich die konkrete Zielsetzung in entscheidenden Details. Die beschriebenen Lösungsansätze sind daher nicht direkt für die Erreichung der hier verfolgten Ziele anwendbar: (1) Um den vorgeschlagenen modellbasierten Prozess durchzuführen, sind zur manuellen Anreicherung der SysML-Modelle mit den vorgeschlagenen HLA-spezifischen UML-Profilen weiterhin simulations- und technologiespezifische Kenntnisse erforderlich. Die hier vorgestellten Lösungskonzepte sehen vor, dass im gesamten Prozess für den Modellentwickler und den Tester keine Kontaktpunkte mit simulations- und technologiespezifischen Strukturen und Mechanismen existieren. (2) Die Implementierung der eigentlichen Logik der Simulationsteilnehmer ist, wie es in MDA üblich ist, den mehrstufigen Modelltransformationen nachgelagert. Bei Verwendung von Szenarien, wie sie in dieser Arbeit verstanden werden, entstünde so die Notwendigkeit, die Verhaltenslogik für jedes Szenario erneut in die generierten Gerüste zu integrieren. Der hier vorgeschlagene Ansatz verlagert diese Tätigkeit im Gesamtprozess weiter nach vorn, sodass zur Spezifikation einer Szenarioinstanz lediglich vorgefertigte Modellbestandteile ausgewählt, zusammengestellt und parametrisiert werden müssen. (3) Die vorgeschlagene Toolkette enthält eine nicht unerhebliche Anzahl unterschiedlicher Software-Anwendungen und -Tools zur Durchführung der einzelnen Prozessschritte, wie die proprietäre Software *Pitch Developer Studio*[67]. Das Ziel der vorliegenden Arbeit ist es auch, solche Abhängigkeiten deutlich zu reduzieren, um die Einstiegshürde gering zu halten. (4) Die vorgestellte Implementierung scheint dem Entwickler keine einfache Möglichkeit zu bieten, innerhalb der zu integrierenden Logik die Daten (Objekte und Attribute) zu nutzen, die von anderen Simulationsteilnehmern zur Verfügung gestellt werden. Zumindest aber bleibt diese Frage offen, da die veröffentlichten Artikel nicht näher darauf eingehen. Auch hier wäre der Entwickler mit technologiespezifischen Details konfrontiert. Weil sich die Grundideen der vorgestellten existierenden Lösungskonzepte trotz der identifizierten konzeptuellen Abweichungen im Kern mit denen dieser überschneiden, flossen wertvolle, aus den genannten Arbeiten gewonnene Erkenntnisse bezüglich einzelner Teilbereiche in die weitere Ausgestaltung der Konzepte und der experimentellen Umsetzung ein.

14.3 INHALTLICHE AUSGESTALTUNG

Nachdem die erarbeiteten Lösungsansätze in Abschnitt 14.1 auf einer abstrakten, technologieunabhängigen Betrachtungsebene vorgestellt wurden, werden nachfolgend zentrale Konzeptbestandteile inhaltlich und funktional näher betrachtet. Aufgrund dessen, dass zentrale Bestandteile der vorgestellten Lösungskonzepte die Möglichkeit einer verteilten Ausführung (vgl. Abschnitt 14.1.1), eine agentenbasierte Struktur sowie entsprechende Abläufe sind, hat die zugrundeliegende Simulationstechnologie zu einem erheblichen Grad Einfluss auf alle weiteren konkreten Entwurfsentscheidungen. Zunächst werden daher einige Standards und Frameworks vorgestellt, die als mögliche Optionen identifiziert wurden, und anschließend wird die konkrete Entscheidung für eine der Optionen begründet.

14.3.1 TECHNOLOGIEAUSWAHL: VERTEILTE AUSFÜHRUNG

Die physikalisch verteilte Ausführung miteinander kommunizierender Prozesse und, als Spezialisierung dessen, auch die verteilte Ausführung von Computersimulationsläufen sind bereits seit den

[67] *Pitch Developer Studio*, Pitch Technologies: `https://onearc.com/products/pitch-developer-tools/` [letzter Zugriff: 26.09.25]

1970er Jahren Gegenstand intensiver Entwicklungs- und Forschungstätigkeiten [Fujimoto, 2003]. Die technologischen Ansätze und die daraus hervorgegangenen konkreten Artefakte sind somit bereits ausgereift und weitverbreitet. Eine vollumfängliche Eigenentwicklung stellt daher heutzutage in den allermeisten Fällen keine vertretbare Option dar. Einige der aktuellsten und geläufigsten Standards und Frameworks wurden deshalb auf ihre Eignung als technologischer Unterbau für die hier entworfene Lösung untersucht. Nachfolgend werden zunächst vier vorgestellt, um anschließend die Entscheidung für einen oder eines dieser nachvollziehbar zu begründen.

Functional Mock-up Interface

Das *Functional Mock-up Interface* (FMI)[68] ist ein offener, toolunabhängiger Standard für den Austausch und die Kopplung dynamischer Modelle. Der Standard wurde im Rahmen des Forschungsprojektes *MODELISAR*[69] entworfen und umgesetzt. Das Ziel war dabei die Optimierung des Designprozesses von Systemen, die mit eingebetteter Software ausgestattet sind. Der Fokus lag dabei auf der Anwendung in der Automobilindustrie. [Awais et al., 2013]

Im Zentrum von FMI stehen die sogenannten *Functional Mock-up Units* (FMU). Eine FMU repräsentiert jeweils ein Teilsystem, genauer gesagt dessen Modell, als eigenständiges Artefakt. Um Austauschbarkeit zu gewährleisten, haben FMUs dabei die Form von ZIP-Dateien, die jeweils (1) eine standardisierte Schnittstellenbeschreibung (in der Regel in Form eines XML-Manifests) zur eindeutigen Definition von Eingabeschnittstellen, Ausgabeschnittstellen und Parametern, (2) eine ausführbare Implementierung des jeweiligen Modells und (3) weitere optionale Dateien enthalten.

FMI ist ausgerichtet auf die numerische Lösung gleichungsbasierter mathematischer Modelle [Blochwitz et al., 2012]. FMUs stellen somit beschreibende Verhaltensmodelle dar (vgl. Abbildung 27a) und die Güte der Ergebnisse ist abhängig vom verwendeten Gleichungslöser (engl.: Solver). Entsprechend des zentralen numerischen Charakters geschieht die Kopplung von FMUs anhand der für jede FMU vordefinierten Ein- und Ausgänge, über welche einzelne Instanzen primitiver Datentypen ausgetauscht werden. Ein Austausch von komplexen, zusammengesetzten Datentypen ist nicht vorgesehen.

Der Standard bietet verschiedene Schnittstellentypen für den Einsatz der gekapselten Modelle: *Model Exchange* (ME) und *Co-Simulation* (CS) [Hatledal, 2021]. Mit der Version 3.0 [Junghanns et al., 2021] wurde zusätzlich der Typ *Scheduled Execution* (SE) eingeführt. Während in der ME-Variante ein zentraler Gleichungslöser vorgesehen ist, enthält bei FMI-CS jede FMU zusätzlich zu den zuvor genannten Inhalten eine eigene Gleichungslöserinstanz (vgl. Abbildung 64). FMI-CS ermöglicht so die direkte Kopplung und gemeinsame, reproduzierbare simulative Ausführung heterogener Teilmodelle, die mit verschiedenen Tools entwickelt wurden, von verschiedenen Anbietern stammen oder unterschiedliche Technologien einsetzen.

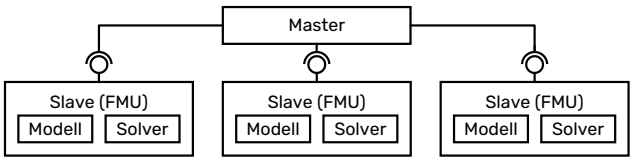


Abbildung 64: Aufbau einer Co-Simulation nach dem FMI-CS-Standard

[68] Modelica Association, *Functional Mock-up Interface*: `https://fmi-standard.org/` [letzter Zugriff: 17.02.26]

[69] Information Technology for European Advancement (ITEA), *Project · 07006 MODELISAR*: `https://itea4.org/project/modelisar.html` [letzter Zugriff: 17.02.26]

FMI-CS folgt bei der Ausführung einem Master-Slave-Paradigma: Jede FMU kapselt ihren eigenen numerischen Lösungsmechanismus und schreitet, gesteuert durch einen externen Master-Algorithmus, bei Bedarf mit ihrem internen Zustand fort. Der Master-Algorithmus ist dabei verantwortlich für (1) die Instanziierung und Initialisierung der FMUs, (2) die Koordination des Fortschreitens der Simulationszeit und (3) die direkte Kommunikation mit den FMUs zum Datenaustausch über die definierten Ein- und Ausgabeschnittstellen. Ein typischer Co-Simulationszyklus verläuft wie folgt: Zur Simulationszeit t setzt der Master die Eingabewerte jeder FMU, löst eine Fortrechnung um einen Zeitschritt h aus und ruft anschließend die von der jeweiligen FMU produzierten Ausgabewerte ab. Aus numerischer Sicht werden Genauigkeit und Stabilität daher bestimmt durch die Wahl von h, das Kopplungsschema (z. B. Jacobi-Verfahren oder Gauß-Seidel-Verfahren) und den Umgang mit Diskontinuitäten. Obgleich der zentralen Rolle des Master-Algorithmus spezifiziert FMI zwar die Komponentenschnittstelle, enthält jedoch keine exakten Vorgaben für eine konkrete Umsetzung eines solchen und schreibt keine universell anwendbaren Orchestrierungs- und Koordinierungsstrategien vor.

FMI-CS wird häufig eingesetzt, wenn die primären Anforderungen an die Simulationsanwendung im Bereich kleinteiliger Interoperabilität und Portabilität numerischer Modelle verordnet sind. Zu den relevantesten Vorteilen gehören dabei unter anderem:

- Die Paketierung und die klare Beschreibung der Schnittstellen über Ein- und Ausgaben reduzieren den Integrationsaufwand bei der Kopplung von Modellen aus unterschiedlichen Umgebungen.
- Die ausführbaren Implementierungen können als Binärdateien vorliegen, was Austausch und Zusammenarbeit ohne Offenlegung des zugrundeliegenden Quellcodes ermöglicht.
- FMUs bieten eine klar definierte Komponentengrenze.
- FMI-CS passt gut zu modellbasierten Entwicklungsprozessen, bei denen Teilsystemteams gepackte Komponenten für die Modellintegration und -verifizierung liefern.

Herausforderungen bezüglich des Einsatzes von FMI-CS sind hingegen u. a.:

- Die Güte der Simulationsergebnisse und die Robustheit des Simulationsablaufs sind abhängig von dem konkret gewählten Master-Algorithmus, dem gewählten Verfahren zur näherungsweisen Lösung und dessen konkreter Umsetzung, der Scheduling-Richtlinie und der Ereignisbehandlung.
- Begrenzte Berücksichtigung von Aspekten verteilter Simulationsausführungen, wie das Fehlen standardisierter Möglichkeiten für eine geografisch verteilte Ausführung.
- Durch den Fokus auf primitive Datentypen können Systeme, die reich an eingebetteten Steuersystemen sind und/oder sich physikalisch direkt beeinflussen, optimal abgebildet werden. Die Abbildung größerer Verbünde kooperierender Gesamtsysteme wird hingegen erschwert.
- Die Versionen 1 und 2 des FMI-Standards sahen keine Mechanismen zur Synchronisation des Fortschreitens der Simulationszeit der einzelnen FMUs vor. Mit Version 3.0 des Standards wurden zusätzlich sogenannte *Timer* und *Clocks* eingeführt [Hansen et al., 2022], welche zur Synchronisierung des Fortschreitens der an einer Co-Simulation teilnehmenden FMUs genutzt werden können. Das Handling stellt sich in diesem Zusammenhang allerdings als kleinteilig und komplex dar.

Der FMI-Standard insgesamt und auch der Schnittstellentyp CS im Speziellen haben seit der Veröffentlichung der ersten Version vorwiegend im Kontext der Simulation von cyber-physischen Systemen und Systemen mit einer Vielzahl von eingebetteten Teilsystemen, insbesondere in der Automotive-Branche, weite Verbreitung gefunden. An dieser Stelle sei beispielhaft auf drei Arbeiten verwiesen: In

[Liu et al., 2024c] wird eine FMI-basierte Simulation der Bewegungsplanung und -durchführung eines auf einem Schiff montierten Ladekrans durchgeführt. In [Bücs et al., 2015] werden mehrere Ansätze für Hardware-in-the-Loop (HiL) Simulationsläufe zum Testen von Multi-Domänen Automotive-Systemen vorgestellt und erprobt. In [Cho et al., 2020] wird ein Framework vorgestellt, welches zum Ziel hat, simulative Testläufe von Fahrsystemen kosteneffizienter durchführen zu können. Zu diesem Zweck wird auf eine Kombination aus FMI und Cloud Computing gesetzt.

Aufgrund der Popularität, die der Standard erreichen konnte, sind ebenfalls einige Frameworks entstanden, die darauf abzielen, bestehende Herausforderungen durch zusätzliche Funktionalitäten zu überwinden und so den Einsatz insgesamt einfacher und effizienter zu gestalten: *Daccosim NG*[70] ist ein Framework und eine Umgebung für die Entwicklung und die Ausführung von kooperativen Simulationen, basierend auf der Java Bibliothek *JavaFMI*[71]. Der Fokus liegt auf der Modellierung und Simulation komplexer maximal 1-dimensionaler Systeme, wie großer cyber-physischer Systeme, Kontroll- und Steuersysteme sowie multidisziplinärer Systeme. [Évora Gómez et al., 2019] *Maestro 2*[72] ist ein Framework bzw. ein Orchestrierungs-Werkzeug für FMI-basierte Co-Simulationen, das zum Ziel hat, durch den Einsatz einer domänenspezifischen Sprache den Einsatz von FMUs in Bezug auf die Simulationsgranularität flexibler zu gestalten und somit die Wiederverwendbarkeit dieser weiter zu erhöhen. [Hansen et al., 2024] *Open Simulation Platform* (OSP)[73] ist eine Open-Source-Initiative einiger relevanter Industrievertreter zur Co-Simulation von maritimen Komponenten und Systemen. Ziel der Entwicklung war es, die Wiederverwendung von Modellen organisationsübergreifend zu ermöglichen, ohne dabei sensible Daten preiszugeben und intellektuelles Eigentum offenzulegen. Dafür setzt das Framework in großem Umfang auf die bestehenden Kernaspekte von FMI und reichert diese um eine Master-Algorithmus-Implementierung in Form einer C/C++-Bibliothek, eine Schnittstellenspezifikation für Modelle von maritimen Komponenten und Systemen sowie einige Referenzmodelle an [Smogeli et al., 2020].

Distributed Co-Simulation Protocol

Das noch junge *Distributed Co-simulation Protocol* (DCP)[74] ist im Umfeld von FMI entstanden und stellt einen speziellen, Plattform- und Übertragungsmedium-unabhängigen Standard zur Kopplung von Modellen und Hardwaresystemen dar. Der Standard konzentriert sich auf die Kommunikationsschicht, um nicht-funktionale Interoperabilität zu erreichen, insbesondere für Echtzeitsysteme. Der verfolgte Ansatz ist ein Teststand-orientierter und legt viel Wert auf die Initialisierung, Konfiguration und Steuerung aller Simulationsteilnehmer durch eine zentrale Master-Instanz.

Die DCP-Spezifikation umfasst ein Datenmodell, eine Zustandsmaschine und ein Kommunikationsprotokoll. Dabei spezifiziert der Standard lediglich die Definition eines Slaves, nicht aber die des zentralen Masters. Aufgrund des Fokus auf die Koordination der Simulationsteilnehmer und Datenübertragung auf Protokollebene wird DCP selten alleine eingesetzt: Häufiger sind Kombinationen mit anderen Technologien, die weitere Funktionalitäten und Aspekte abdecken, wie FMI oder HLA.

[70] Bitbucket - DACCOSIM NG Wiki, simulage: `https://bitbucket.org/simulage/daccosim` [letzter Zugriff: 17.02.26]
[71] Bitbucket - JavaFMI Wiki, siani: `https://bitbucket.org/siani/javafmi` [letzter Zugriff: 17.02.26]
[72] GitHub - maestro, INTO-CPS-Association: `https://github.com/INTO-CPS-Association/maestro` [letzter Zugriff: 17.02.26]
[73] *Open Simulation Platform*, DNV AS: `https://opensimulationplatform.com/`[letzter Zugriff: 17.02.26]
[74] *Distributed Co-Simulation Protocol Project*, Modelica Association: `https://dcp-standard.org/` [letzter Zugriff: 17.02.26]

DCP ist eine vielversprechende Lösung für die Realisierung von Digital-Twin-Anwendungen und die Kopplung von Hardware-Einheiten, die eine Echtzeit-Interaktion mit physischen Komponenten erfordern [Hatledal, 2021]. Allerdings ist der Standard noch jung und scheint bisher keine weit verbreitete Anwendung gefunden zu haben. Aufgrund dessen sowie der hardwarenäheren Ausrichtung, als es das hier vorgestellte Konzept erfordert, wird DCP an dieser Stelle nicht weiter in Betracht gezogen.

MOSAIK

Das seit 2021 in der dritten Version verfügbare Open-Source-Framework *MOSAIK*[75] fungiert hauptsächlich als Integrator und Koordinator für Co-Simulationen. Es ermöglicht die Wiederverwendung und Kopplung bestehender Simulationsanwendungen und Modelle zur Ausführung integrierter koordinierter Szenarien. Der Fokus liegt dabei auf der Simulation von cyber-physischen Energiesystemen. Das Framework bietet zur Umsetzung (1) eine Szenario-API zum Erstellen eines Szenarios, (2) eine Komponenten-API zur Kommunikation mit MOSAIK, von welcher je eine Implementierung in jedem Simulationsteilnehmer integriert oder diesem vorgeschaltet sein muss, und (3) einen Scheduler, der die Ausführungsreihenfolge koordiniert und den Datenaustausch realisiert. [Steinbrink et al., 2018]

Um den Datenaustausch zwischen den Simulationseinheiten zu konfigurieren, werden in den API-konformen Szenarien beschreibende Entitäten instanziiert und Attribute dieser miteinander verbunden, um sogenannte Data-Flows zu schaffen. Zur Laufzeit werden die entsprechenden Ausgangswerte von MOSAIK entgegengenommen und entsprechend der konfigurierten Data-Flows beim Fortschreiten der Simulationszeit an die Eingänge anderer Teilnehmer geliefert. Dabei wird zeitdiskretes, ereignisbasiertes und sogenanntes hybrides Fortschreiten unterstützt. [Ofenloch et al., 2022]

High-Level Architecture (HLA)

Der älteste hier genauer betrachtete Ansatz ist ohne Zweifel die *High-Level Architecture* (HLA): Bereits in den frühen 1990er Jahren begann die Entwicklung, initiiert durch das Verteidigungsministerium der Vereinigten Staaten (US Department of Defense). Die erste öffentlich zugängliche Version (1.3) wurde anschließend im Jahr 1998 herausgegeben. Kurze Zeit später wurde das gesamte Projekt an den Berufsverband *Institute of Electrical and Electronics Engineers* (IEEE) übergeben, welcher im Jahr 2000 den ersten entsprechenden internationalen Standard unter dem Namen *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules* herausbrachte. Zehn Jahre später wurde eine weiterentwickelte Version unter dem Namen *HLA Evolved* veröffentlicht [IEEE, 1516:2010]. Trotz des Alters des Frameworks wird auch weiterhin an dessen Verbesserung und Anpassung an sich verändernde Anforderungen gearbeitet[76]. [Gütlein, 2023]

Der Standard umfasst eine Reihe von Dokumenten, welche unterschiedliche Aspekte der definierten offenen Architektur und ihrer Schnittstellen für den Aufbau verteilter und interoperabler Simulationsanwendungen dokumentieren:

- *Framework and Rules* [IEEE, 1516:2010] beschreibt zehn klar definierte Regeln, die von den einzelnen Simulationsteilnehmern, welche im HLA-Kontext als *Federate* (dt.: Föderationsmitglied) bezeichnet

[75] *mosaik - A flexible Smart Grid co-simulation framework*, OFFIS e.V.: `https://mosaik.offis.de/` [letzter Zugriff: 17.02.26]

[76] In der Mitte des Jahres 2025 wurde die aktuellste, als *HLA 4* bezeichnete, Iteration unter der Nummer *1516-2025* veröffentlicht. Auf Grund der zu diesem Zeitpunkt bereits sehr weit fortgeschrittenen Umsetzung des hier vorgestellten softwaretechnischen Prototyps, wird diese aber im Rahmen dieser Arbeit nicht weiter betrachtet.
Der Standard ist unter folgender Adresse abrufbar: `https://standards.ieee.org/ieee/1516/6687/` [letzter Zugriff: 17.02.26]

werden, und der zusammengesetzten Gesamtsimulation, entsprechend als *Federation* (dt.: Föderation) bezeichnet, zwingend einzuhalten sind.
- Die *Federate Interface Specification* [IEEE, 1516.1:2010] spezifiziert die von der koordinierenden Einheit, die als *Runtime Infrastructure (RTI)* (dt.: Laufzeitinfrastruktur) bezeichnet wird, bereitzustellenden Dienste und die entsprechenden Nutzungsschnittstellen.
- Die *Object Model Template Specification* [IEEE, 1516.2:2010] spezifiziert das Format der HLA-eigenen Datenstruktur, die für den Austausch von Daten zwischen den Federates genutzt wird, in Form des *Object Model Template (OMT)*.

Das Ziel der Dokumente ist es, eine präzise und ausführliche Architekturbeschreibung zu liefern, deren Umsetzung die Wiederverwendbarkeit und Interoperabilität von Simulationsmodellen in physikalisch verteilten Anwendungsfällen ermöglicht.

Der Standard definiert die Dienste und Schnittstellen, die zur Realisierung von verteilten Simulationsanwendungen durch die Federates und die RTI bereitgestellt werden müssen, auf einer konzeptionellen Ebene. Es existiert daher, wie es auch bei FMI und DCP der Fall ist, keine Referenzimplementierung. Auch gibt es bezüglich des zugrundeliegenden Kommunikationsprotokolls keine konkreten Vorgaben und die Wahl bleibt der Implementierung überlassen.

Während eines Simulationslaufs werden drei Arten von Datenmodellen genutzt, die alle auf dem OMT basieren: das *Simulation Object Model* (SOM) (dt.: Simulationsobjektmodell), das *Federation Object Model* (FOM) (dt.: Föderationsobjektmodell) und das *Management Object Model* (MOM) (dt.: Verwaltungsobjektmodell) (vgl. Abbildung 65). Das SOM bezieht sich auf ein individuelles Federate und beschreibt die Schnittstellen desselben. Genauer enthält es Informationen über die möglichen Datenstrukturen, die das Federate veröffentlichen (*publish*) oder konsumieren (*subscribe*) kann. Das FOM definiert alle Informationstypen, die während des Simulationslaufs der Federation zwischen den Federates ausgetauscht werden können (u. a. Objekte und ihre Attribute) [Gütlein et al., 2020]. Das MOM ist fester Bestandteil des Standards und somit jeder Implementierung und enthält Informationen zu möglichen Steuerungs- und Managementaktionen, die von der RTI durchgeführt werden können. Die Definition der HLA-Datenaustauschmodelle hat mit der Veröffentlichung der Version IEEE 1516.2-2010 eine wichtige Erweiterung erhalten: Die Definition von FOMs folgt seitdem einem modularen Ansatz und jedes Federate kann sein eigenes FOM in die Federation einbringen, welche dann von der RTI zur Laufzeit zusammengeführt werden [Gütlein et al., 2020; Möller et al., 2007].

Die OMT-konformen Datenstrukturen bilden Klassen möglicher Objekte ab. Objektklassen bestehen dabei jeweils aus einer Menge von Attributen. Jede Objektinstanz ist während des Simulationslaufs zu einem bestimmten Zeitpunkt durch ihren Zustand definiert, der durch die aktuellen Attributwerte spezifiziert wird. Das für die Verwaltung eines bestimmten Objekts zuständige Federate kann den Zu-

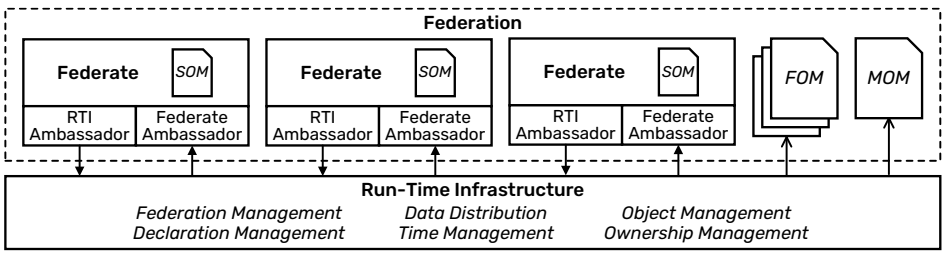


Abbildung 65: Aufbau einer verteilten Simulation nach dem HLA-Standard

stand der Objektinstanz durch Anpassung der Attributwerte ändern (*updating*). Dafür werden Dienste der RTI genutzt. Anschließend verteilt die RTI die Aktualisierung an alle Federates, die Interesse an dieser Objektklasse und diesem Attribut angemeldet haben (*subscribe*). Die vom OMT vorgegebene Datenstruktur ermöglicht dabei die Spezifizierung von Objektklassen in Hierarchien, ähnlich klassischer Vererbung, wie sie in der objektorientierten Programmierung zum Einsatz kommt.

Um Federation-weite Abläufe wie die o. g. Datenverteilung koordiniert zu ermöglichen, sind für eine Implementierung insgesamt sechs simulationsverwaltende und -steuernde Dienstgruppen durch Standard vorgeschrieben (vgl. Abbildung 65):

Federation Management umfasst Funktionalitäten für das Erstellen, Initialisieren und Löschen einer Federation sowie die dynamische Steuerung und Durchführung von etwaigen Änderungen während eines Simulationslaufs. [IEEE, 1516.1:2010]

Declaration Management umfasst Dienste, die von Federates genutzt werden können, um zur Simulationslaufzeit zu deklarieren, welche Objekte/Objektklassen und Attribute des FOM sie veröffentlichen (*publish*) und/oder abonnieren (*subscribe*) möchten. Dafür werden Dienstschnittstellen wie *PublishObjectClassAttributes(objectClass, attributes)* zur Verfügung gestellt. [IEEE, 1516.1:2010]

Object Management stellt Dienste zur Verfügung, welche von Federates genutzt werden können, um neue Objektinstanzen anzumelden sowie ihnen zugeordnete bestehende Objektinstanzen zu löschen oder zu modifizieren. Die entsprechenden Änderungen werden daraufhin an alle anderen interessierten Federates weitergeleitet. [IEEE, 1516.1:2010]

Ownership Management stellt Dienste zur Verfügung, die von den Federates genutzt werden können, um das *Ownership* (dt.: Eigentumsverhältnis) konkreter Objektinstanzen an ein anderes Federate zu übertragen. Dadurch wird es ermöglicht, dass mehrere Federates die Wertzuweisungen von Attributen einer Objektklasse gemeinsam übernehmen, denn die Aktualisierung einer Wertezuweisung ist nur dem aktuellen Eigentümer der Objektinstanz erlaubt. [IEEE, 1516.1:2010]

Time Management umfasst Dienste, die u. a. die Zustellung von Nachrichten, wie Benachrichtigungen über Attribut-Updates, in einer bestimmten kontrollierten logischen Reihenfolge innerhalb einer Föderation sicherstellen. Dafür wird eine Federation-zentrale Zeitachse genutzt, auf welcher sich sowohl die Federation (repräsentiert durch die RTI) als auch die einzelnen Federates während eines Simulationslaufs selbstständig vorwärtsbewegen. Die Time Management Dienste stellen dabei sicher, dass dieses Fortschreiten koordiniert geschieht. Ein zentraler Mechanismus dabei sind die *Time Advance Requests (TAR)*, welche von den Federates genutzt werden, um die Erlaubnis zum Fortschreiten auf der Zeitachse zu einem bestimmten Zeitpunkt anzufragen. [IEEE, 1516.1:2010; Fujimoto, 1998]

Data Distribution Management umfasst Dienste, die genutzt werden können, um die Anzahl unnötiger Datenübertragungen zu reduzieren. Dafür können sogenannte *Dimensionen* definiert werden, um Werteräume aufzuspannen. Sowohl die Attribut-besitzenden als auch die Attribut-konsumierenden Federates können anschließend Bereichsangaben innerhalb dieser Werteräume spezifizieren. Durch einen Abgleich dieser, wird dann durch die RTI bestimmt, welche Kommunikation tatsächlich relevant ist. [IEEE, 1516.1:2010; Morse & Steinman, 1997]

Entscheidung

Aufgrund der geringen Überschneidung des funktionalen und inhaltlichen Fokus von MOSAIK mit der Ausrichtung dieser Arbeit wurde das Framework als wenig geeignet eingestuft, die Grundlage für eine Umsetzung der vorgestellten Konzepte zu bilden.

Im Rahmen einer vergleichenden Bewertung lässt sich FMI-CS in erster Linie als standardisierte Schnittstelle für den Austausch und die Kopplung von heterogenen Modellen kleinteiliger Systemkomponenten charakterisieren. Die Stärken kommen vorrangig dann zum Tragen, wenn Systeme getestet werden sollen, welche viele Systemkomponenten umfassen, die aus unterschiedlichen Quellen stammen und direkt miteinander zum Austausch numerischer Werte kommunizieren, und gleichzeitig die Anforderungen an eine physische Verteilung gering sind. FMI zeigt sich somit als eine geeignete Grundlage für die simulative Erprobung interner Wirkzusammenhänge einzelner Fahrzeuge als Kombination einer großen Zahl kleinteiliger, eingebetteter Systeme verschiedener Hersteller und der Umsetzung von digitalen Zwillingen. Zur Abbildung einer von den Simulationsteilnehmern gemeinsam genutzten Umwelt ist FMI hingegen nur bedingt geeignet. Auch eine physikalische Verteilung wird von FMI nicht direkt adressiert. Ebenso deckt sich auch der Fokus auf numerische Modelle nicht mit der Ausrichtung des erarbeiteten Konzepts auf erklärende Verhaltensmodelle. Das zentrale Merkmal von FMI, die Kapselung einzelner ausführbarer Modelle in FMUs, und die damit einhergehenden Vorteile sind im Rahmen dieser Arbeit nur von geringem Interesse, da u. a. die Modelle kein externes, zu schützendes Eigentum beinhalten und auf demselben Metamodell basieren. Der Schutz der in den Simulationslauf eingebundenen externen Funktionseinheiten wird stattdessen durch das Konzept der Proxys ermöglicht (vgl. Abschnitt 14.1.7)

Im Gegensatz dazu zielt der *High-Level Architecture* (HLA)-Standard explizit auf eine vermittlerbasierte, geografisch verteilte, simulative Abbildung von System-of-Systems (SoS) ab. Die standardisierten Laufzeitdienste (insbesondere das Federationmanagement, Zeitmanagement und das Objektmanagement) stellen eine umfassende Grundlage für die angedachten agentenbasierten Funktionalitäten des erarbeiteten Konzepts dar (vgl. Abschnitt 14.1.1). Auch die von HLA durch das OMT vorgeschriebene Datenstruktur hierarchisch geordneter Objektklassen ist näher an der Struktur eines Szenarios und des auf diesem basierenden Simulationslaufs, wie sie in den Abschnitten 14.1.2, 14.1.3 und 14.1.5 konzipiert wurden. Durch die Nutzung des *Publish-Subscribe* Entwurfsmusters, die Konfigurationsflexibilität sowie die Möglichkeit modularer FOMs sind verteilte HLA-konforme Simulationsanwendungen nur lose gekoppelt, was einer flexiblen und dynamischen Nutzung, wie sie für die Teilkonzepte der Abschnitte 14.1.4 und 14.1.7 nötig ist, zusätzlich entgegenkommt.

Ein prominentes und valides Argument gegen den Einsatz von HLA ist die hohe Komplexität der Entwicklung einer HLA-konformen verteilten Simulationsanwendung, wie sie u. a. in [Xu et al., 2021] bemängelt wird. Die hohe Komplexität entsteht hauptsächlich durch den umfangreichen Standard, welcher eine große Anzahl an Diensten, Schnittstellen und Datenstrukturen kleinteilig beschreibt, dabei aber keine Referenzimplementierung mitbringt. Vor allem die Notwendigkeit, die Mechanismen des Datenaustausches und der Zeitsynchronisierung in jede Simulationseinheit programmatisch integrieren zu müssen, macht Entwicklern ohne hinreichende Erfahrung im Umgang mit dem Standard die Umsetzung schwer. Aufgrund dessen, dass das erarbeitete Konzept jedoch ohnehin eine möglichst umfassende Kapselung der internen Abläufe und Strukturen vorsieht, wird der Umfang und die Ko-

mplexität des HLA-Standards keinen Einfluss auf die Tätigkeiten der identifizierten Nutzergruppen *Modellentwickler* und *Tester* haben (vgl. Abschnitt 13.1).

Aus den o. g. Gründen wurde die *High Level Architecture* als technologische Grundlage für die Ausarbeitung der Konzeptdetails und die experimentelle Umsetzung gewählt. Nachfolgend vorgestellte Entwurfsentscheidungen haben somit zumeist einen Bezug zu Abläufen und Strukturen, wie sie in [IEEE, 1516:2010], [IEEE, 1516.1:2010] und [IEEE, 1516.2:2010] definiert sind. Dementsprechend können die nachfolgenden Detailkonzepte nicht mehr als vollständig technologieunabhängig angesehen werden.

14.3.2 METAMODELL

In den Abschnitten 14.1.2 und 14.1.6 wurde das zentrale Lösungskonzept eines komponentenbasierten Metamodells als gemeinsame Grundlage für die technische Spezifikation einer Szenarioinstanz wie auch die simulative Ausführung dieser vorgestellt. Um als Basis für eine entsprechende programmatische Umsetzung zu dienen, muss dieses auf eine detailliertere Ebene gehoben und mit Inhalten gefüllt werden. Die nachfolgenden Unterabschnitte stellen aus diesem Grund separat die Kernbereiche des erarbeiteten Metamodells konkret im Detail vor.

Die Struktur der Inhalte von Szenarien und Simulationsläufen, wie sie in dieser Arbeit betrachtet werden, orientiert sich an abgeschlossenen Entitäten der realen Welt. Diese Betrachtungsweise gleicht dem Grundgedanken der *objektorientierten Programmierung* (OOP). Naheliegenderweise zielen alle nachfolgend vorgestellten Entwürfe und Konzeptinhalte auf eine entsprechende objektorientierte Umsetzung ab, und für die Visualisierungen wurde mehrheitlich auf den Einsatz struktureller Diagrammtypen der *Unified Modelling Language* (UML) zurückgegriffen.

Die nachfolgenden Darstellungen und Beschreibungen sind nicht immer vollständig: Um die Übersichtlichkeit und Lesbarkeit zu wahren, werden lediglich die relevantesten Konzeptteile betrachtet. An dieser Stelle nicht im Detail beleuchtete Zusammenhänge lassen sich aber, unter der Voraussetzung ausreichender Programmierkenntnisse, aus dem öffentlich zugänglichen Programmcode der experimentellen Umsetzung erschließen. Die experimentelle Umsetzung ist Thema des Abschnitts 15.

An dieser Stelle muss etwas vorgegriffen werden, um das Verständnis der nachfolgend dargestellten Modellstrukturen sicherzustellen. Das Metamodell sieht eine abstrakte Wurzelklasse für alle Klassen vor, die Konzepte beschreiben, die unmittelbarer Teil einer Szenarioinstanz sein können: *ScenarioSpecificationItem*. Von dieser müssen entsprechende Klassen direkt oder indirekt durch Vererbung abgeleitet werden. Diese abstrakte Klasse stellt sicher, dass alle Instanzen abgeleiteter Klassen serialisierbar sind und das Attribut *id* besitzen (vgl. Abbildung 66). Zusätzlich ist durch die einheitliche Superklasse sichergestellt, dass beim Einlesen und Interpretieren von Szenarioinstanzen alle enthaltenen Elemente bei Bedarf als Instanzen desselben Typs behandelt werden können. Die eindeutige *id* ist für viele im weiteren Verlauf beschriebenen Funktionalitäten essenziell, wie für das Mapping von, innerhalb einer Federation ausgetauschten, OMT-konformen Daten auf objektorientierte Repräsentationen dieser. Aus Gründen der Übersicht wird die indirekte oder direkte Vererbungsbeziehung zu *ScenarioSpecificationItem* im weiteren Verlauf nicht mehr explizit dargestellt. Stattdessen werden Klassen, die eine un-

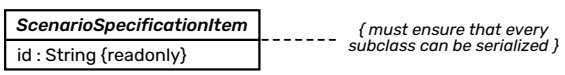


Abbildung 66: Metamodell – abstrakte Superklasse aller möglichen Inhalte einer Szenarioinstanz

mittelbare Subklasse von *ScenarioSpecificationItem* sind, in der Kopfzeile mit Ⓢ markiert und Klassen, die eine indirekte Subklasse von *ScenarioSpecificationItem* sind, analog mit Ⓢ.

14.3.2.1 ELEMENTE

Im Zentrum möglicher Szenarioinhalte steht das Konzept des *Elements* (vgl. Abbildung 47). Ein Element repräsentiert eine für sich stehende Entität als Teil einer Szene. In Abschnitt 7.2.4 wurde *Element* als abstraktes Konzept eingeführt, für das mehrere mögliche konkrete Ausprägungen existieren: Ein *Element* kann *stationär* (unveränderbare Position im Raum) oder *dynamisch* (veränderbare Position im Raum) sein. Nach den Definitionen in Abschnitt 7.2.4, angelehnt an [Ulbrich et al., 2015], können Elemente dabei auch nicht-greifbarer Natur sein, wie eines, das die aktuellen Wetterverhältnisse für den abgebildeten Raum oder einen Teil von diesem beschreibt. Das entworfene Metamodell bildet diese Unterscheidung bezüglich der physischen Eigenschaften eines Elements durch die in Abbildung 67 gezeigte Vererbungshierarchie ab. Ein konkretes Modellelement kann somit auf einer der vier abstrakten Klassen *GlobalNonPhysicalElement*, *LocalNonPhysicalElement*, *StationaryElement*, *DynamicPhysicalElement* aufbauen, die die genannten Eigenschaften jeweils durch die Menge ihrer Felder abbilden.

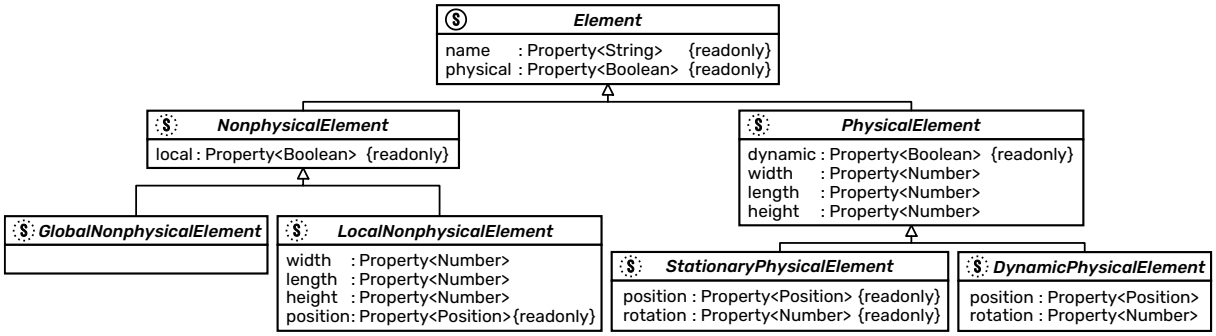


Abbildung 67: Metamodell – Hierarchie physischer Charakteristiken

Die Definitionen in Abschnitt 7.2.4 teilen *Elemente* zusätzlich zu den physischen Eigenschaften in die Rollen *Akteur* und *Beobachter* ein. Jede konkrete kombinatorische Ausprägung der physischen Charakteristiken kann eine der beiden oder beide Rollen einnehmen. Um die Vererbungshierarchien flacher und übersichtlicher zu halten, werden diese verhaltensorientierten Eigenschaften explizit durch das Metamodell abgebildet. Die meisten gebräuchlichen objektorientierten Programmiersprachen lassen indes keine Mehrfachvererbung zu. Da die Ausprägungen physischer Eigenschaften bereits in Form von abstrakten Klassen modelliert wurden, muss für die Ausprägungen verhaltensbezogener Charakteristiken daher auf das Konzept der *Schnittstelle* (engl.: interface) zurückgegriffen werden. In Interfaces ist es in der Regel wiederum nicht möglich, Instanzvariablen zu definieren. Diese Einschränkung wird bestmöglich umgangen, indem die verhaltensdeklarierenden Interfaces statt des Feldes selbst, wie es bei einer abstrakten Klasse der Fall wäre, jeweils entsprechende *get*- und *set*- Methoden vorschreiben. Zur Wahrung der Übersicht wird in den hier gezeigten Klassendiagrammen trotzdem von Interfaces auf andere Klassen durch Assoziationen und Kompositionen verwiesen. Diese sind beim Lesen der Diagramme gedanklich durch entsprechende *set*- und *get*-Methoden zu ersetzen.

Abbildung 68 zeigt die vier Interfaces des Metamodells zur Abbildung der möglichen verhaltensbezogenen Eigenschaftsausprägungen. Zusätzlich zu den genannten Ausprägungen Beobachter (engl.: observer) und Akteur (engl.: actor) wurden zusätzlich *Prop* (dt.: Requisite) als Markierungsinterface für

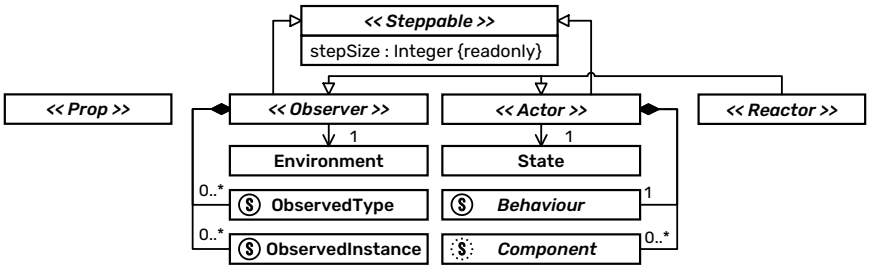


Abbildung 68: Metamodell – verhaltensbezogene Charakteristiken

Elemente ohne zur Laufzeit aktive Funktionsausführung sowie *Reactor* (abgeleitet von *react*, reagieren, in Anlehnung an *Actor* als etwas, das agiert) als Kombination von *Actor* und *Observer* eingeführt.

Ein *Observer* referenziert eine Menge *ObservedType*- und *ObservedInstance*-Objekte, die der Spezifikation von zu abonnierenden Daten anderer Simulationselemente dienen (*subscribe*, vgl. Abschnitt 14.3.1). Ein *Actor* umfasst genau ein *Behaviour* (dt.: Verhalten) zur Abbildung von dynamischem Verhalten und eine variable Menge von *Components* (dt.: Komponenten). Der Aufbau und die Funktionsweise der Klasse *Behaviour* werden in den nachfolgenden Abschnitten *Verhalten* und *Modelltransformation* im Detail betrachtet. Der Aufbau einer Komponente wird im Abschnitt „Komponenten“ beleuchtet, ebenso die mit den Komponenten verbundenen Funktionalitäten im Abschnitt „Modelltransformation“.

Die Klassen *Environment* und *State* repräsentieren die (System-)Umgebung und den internen Zustand des Elements im Sinne des üblichen Aufbaus als Agent (vgl. Abschnitt 7.3.3.2). Sowohl *Environment* als auch *State* wurden dafür als zentrale Dienstklassen (engl.: service) entworfen, die, angelehnt an das Entwurfsmuster des *Einzelstücks* (engl.: singleton), pro Element nur einmal instanziiert werden und allen nötigen Klassen eine öffentliche Schnittstelle für das Hinzufügen, Verändern und Abrufen von Informationen zur Verfügung stellen. Die Funktionsweise der beiden Klassen wird im nachfolgenden Entwurfsabschnitt „Simulation“ genauer betrachtet.

Mittels Kombination von Klassen- und Schnittstellenvererbung der vier Ausprägungen physischer Elementcharakteristiken und der vier Ausprägungen verhaltensbezogener Charakteristiken werden die elf in Tabelle 14 dargestellten Basisklassen für die Umsetzung konkreter Elemente durch das Metamodell zur Verfügung gestellt. Bei zehn der elf Basisklassen handelt es sich wiederum um abstrakte Klassen, die vor einer möglichen Nutzung in einer Szenarioinstanz zunächst ableitend konkretisiert werden müssen. Lediglich *GlobalObserver* steht als konkrete, d. h. instanziierbare, Klasse zur Verfügung. Der *GlobalObserver* stellt dabei ein nicht-physisches globales Beobachter-Element dar und bedient somit die Forderung nach Beobachtbarkeit des Simulationslaufs.

Tabelle 14: Metamodell – kombinierte Ausprägungen der physischen und verhaltensbezogenen Charakteristiken

	NonphysicalElement		***PhysicalElement***	
	Global	***Local***	***Stationary***	***Dynamic***
Prop	-	-	*StationaryProp*	-
Observer	GlobalObserver	*LocalObserver*	-	-
Actor	*GlobalActor*	*LocalActor*	*StationaryActor*	*DynamicActor*
Reactor	*GlocalReactor*	*LocalReactor*	*StationaryReactor*	*DynamicReactor*

14.3.2.2 KOMPONENTEN

Entsprechend des erarbeiteten Lösungskonzeptes eines komponentenbasierten Szenario- und Simulationsmodells (vgl. Abschnitt 14.1.2) können Akteure eine beliebige Menge Komponenten (engl.: component) umfassen. Die entsprechende Metamodell-Klasse *Component* ist in Abbildung 69 dargestellt. *Component* ist dabei ebenfalls von *Element* abgeleitet und implementiert die Schnittstelle *Actor*. Entsprechend hält eine instanziierte *Component* zur Laufzeit eine Referenz zum zentralen *State* des Eltern-Akteurs und eine eigene Verhaltensbeschreibung. Eine *Component* kann somit ebenfalls dynamische Veränderungen am State bewirken.

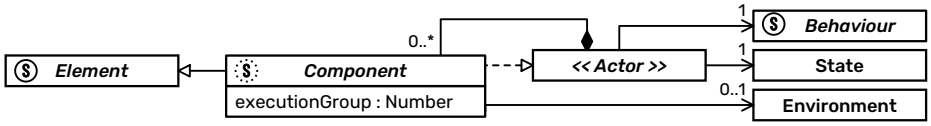


Abbildung 69: Metamodell – Komponente

Zusätzlich kann eine Komponente unter der Voraussetzung, dass das Wurzel-*Element* vom Typ *Reactor* ist, eine Referenz auf die zentrale Umgebungsrepräsentation halten und nutzen. Durch die von *Actor* geerbte Fähigkeit einer *Component*, selbst eine beliebige Menge *Component*-Referenzen vorzuhalten, lassen sich Komponentenhierarchien abbilden, wie sie u. a. in Abbildung 45 zu sehen sind. Zur Realisierung der Abbildbarkeit sowohl von parallel als auch seriell arbeitenden Gruppen von Systemkomponenten besitzt die *Component*-Klasse das Attribut *executionGroup*, mit welchem der Platz in der Ausführungsreihenfolge der Ebene im Komponentenbaum, auf welcher sich die Komponente befindet, festgelegt wird. Der Mechanismus zur Ausführung der Verhaltensbeschreibungen aller Komponenten eines Wurzel-*Elements* in der in einer Szenarioinstanz definierten Reihenfolge wird im nachfolgenden Abschnitt *Modelltransformation* beschrieben. Die feingranulare Konfiguration der Ausführungsreihenfolge in Kombination mit dem möglichen Zugriff auf die *Environment* und den *State* ermöglicht u. a. die Abbildung von Sensoren, wie es der Agentenbegriff vorsieht (vgl. Abschnitt 7.3.3.2): Das *Behaviour* von Komponenten, die in der Ausführungsreihenfolge weit vorn verordnet sind, kann zu Beginn eines Zeitschritts Informationen aus der *Environment* ableiten und in den *State* schreiben.

14.3.2.3 ZUSTANDSVARIABLEN UND AUSGANGSGRÖßEN

Im Rahmen der explorativen Phase wurde in Abschnitt 6.3 *Systembegriff* festgestellt, dass der aktuelle Zustand eines (Teil-)Systems sich durch die Menge seiner Zustandsvariablen und ihre aktuelle Wertebelegung definiert. Zur Modellierung der Zustandsvariablen im systemtheoretischen Sinn bietet das Metamodell die generische Klasse *Property* (vgl. Abbildung 70). Jedes *Element* kann eine beliebige Anzahl an *Properties* besitzen, welche die von außerhalb des Elements wahrnehmbaren Variablen repräsentieren. Die Klasse *Property* bietet so ebenfalls die Grundlage für die Verteilung und Nutzung der Attribute innerhalb einer HLA-konformen Simulation: Das Attribut *publish* wird genutzt, um festzulegen, ob die *Property,* genauer gesagt der ihr zugewiesene Wert (*value),* unter Nutzung der RTI-Dienste aus der Gruppe *Declaration Management* für die Federation als Ausgangsgröße veröffentlicht werden soll (vgl. Abschnitt 14.3.1, *High Level Architecture*). Entsprechend werden nur Objektinstanzen und Attribute über das *Object Management* angemeldet, bei denen *publish* den Wert *true* hat.
Um die Interoperabilität sowohl der simulierten Elemente untereinander als auch in erster Annäherung mit externen Testobjekten (vgl. Abschnitt 14.1.6) zu wahren, hat jede Property zusätzlich jeweils

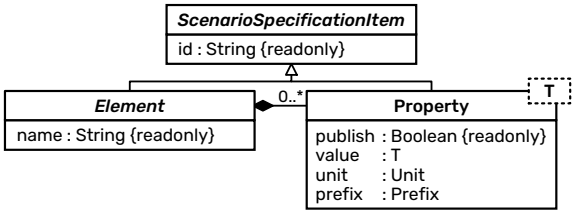


Abbildung 70: Metamodell – Basiselemente *ScenarioSpecificationItem*, *Element* und *Property*

ein Attribut zur Beschreibung der Maßeinheiten (*unit*) sowie eines entsprechenden Einheitenvorsatzes (*prefix*). Diese beiden Angaben können zur automatisierten Umrechnung numerischer physikalischer Werte genutzt werden und bilden somit die Grundlage für das Dictionary, wie es in Abschnitt 14.1.7 konzeptionell eingeführt wurde. Der Aufbau der entsprechenden Klassen *Unit*, *UnitQuantity* und *Prefix* ist in Abbildung 71 dargestellt. Die Bedeutung der dargestellten Attribute sowie der Ablauf der Übersetzung von Properties unterschiedlicher Einheiten und Präfixe sind Betrachtungsgegenstand des nachfolgenden Abschnitts „Bidirektionale Informationstransformation".

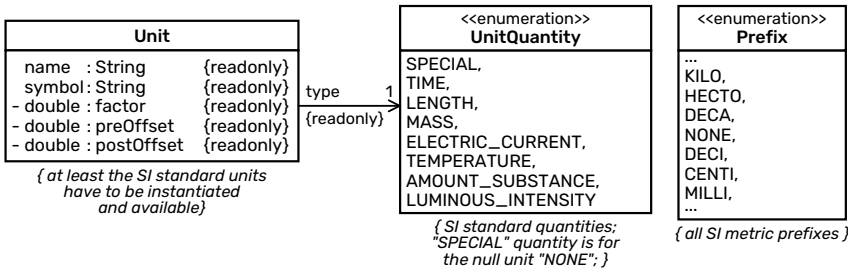


Abbildung 71: Metamodell – *Unit* und *Prefix* als Repräsentation des Internationale Einheitensystems bilden die Grundlage für die Umrechnungen unterschiedliche physikalischer Größen derselben Dimension

Weil *Property* generisch typisiert ist, sind als Wert zusätzlich zu den üblichen primitiven Datentypen, genauer gesagt Instanzen entsprechender Wrapper-Klassen, auch Objekte anderer Typen zulässig. Dies ermöglicht die Modellierung komplexer zusammengesetzter Daten. Das Metamodell selbst umfasst bereits die Klasse *Position* (vgl. Abbildung 72), welche einen zusammengesetzten Datentyp darstellt: Eine Objektinstanz von *Position* ist jedem physischen Element als *value* einer *Property* zugewiesen. Eine *Position* wiederum umfasst selbst Attribute vom Typ *Property* zur Abbildung der konkreten numerischen Werte für jede Dimension im simulierten Raum. Die einzelnen Koordinatenwerte können somit im Rahmen der HLA-basierten Ausführung separat veröffentlicht und abonniert werden.

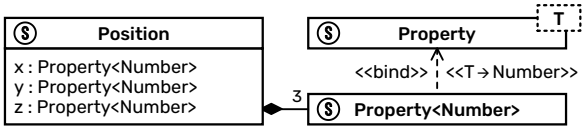


Abbildung 72: Metamodell – *Position* als konkretes Beispiel zusammengesetzter Datentypen

14.3.2.4 EINGANGSGRÖßEN

Ein (Teil-)System hat nach Abschnitt 6.3 *Systembegriff* zusätzlich zu den erwähnten Ausgangsgrößen auch eine Menge an Eingangsgrößen, die Einfluss auf das gezeigte Verhalten des (Teil-)Systems haben können. Analog zu den Ausgangsgrößen, welche sich den *Publishing*-Mechanismus von HLA zunutze machen, werden gewünschte Eingangsgrößen in *Subscription*s überführt. Zur Spezifikation, welche Werte ein simuliertes Element erwartet, können jedem Element, das die *Observer*-Schnittstelle um-

setzt, Mengen von *ObservedInstance*- und *ObservedType*-Objekten zugewiesen werden. Erste referenzieren eine einzelne konkrete Objektinstanz anhand der *id* (vgl. zur zentralen Rolle der *id* die Einleitung des Abschnitts 14.3.2) und die zweitgenannte referenziert Objekttypen anhand des Klassennamens. Beiden gemeinsam sind die Referenzen auf individuelle Eigenschaftswerte über den Namen der *Property*. Mittels zusammengesetzter Namenspfade können dabei ebenfalls Properties komplexer Zustandsvariablen und von Komponenten adressiert werden (vgl. Abschnitt 14.3.3.2).

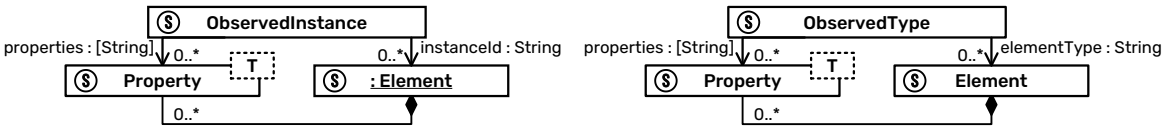


Abbildung 73: Metamodell – Elemente zur Spezifikation der Eingangsgrößen eines Simulationsteilnehmers

14.3.2.5 VERHALTEN

In Abbildung 68 war bereits zu sehen, dass jedem *Actor* eine Verhaltensbeschreibung in Form eines *Behaviours* zugewiesen werden muss. Die abstrakte Klasse *Behaviour* als Basis für Ableitungen höherer Modellschichten schreibt dabei, gemäß dem in Abschnitt 14.1.5.2 vorgestellten Teilkonzept, lediglich die Steuerungsschnittstelle in Form der abstrakten Methode *step* vor (vgl. Abbildung 74), welche von der Simulationssteuerung beim Fortschreiten der Simulationszeit aufgerufen wird. Der Methodenparameter *time* gibt dabei an, um wie viele Zeitschritte vorrangeschritten werden soll. Die bereits konkret vorhandene Methode *init* dient dem Durchreichen von Referenzen zum besitzenden *Actor*, dem zentralen *State* und dem zentralen *Environment* im Sinne des Entwurfsmusters der Abhängigkeitsinjektion (engl.: Dependency Injection).

Die im Abschnitt 14.1.7 vorgestellten Lösungskonzepte, um syntaktische Interoperabilität mit externen Testobjekten zu schaffen, sehen einen *Proxy-Akteur* und eine *Proxy-Komponente* vor. Das Metamodell ist indes so aufgebaut, dass sämtliches dynamisches Verhalten in *Behaviour*-Instanzen gekapselt ist und Akteure und Komponenten als Ableitung von *Element* einen strukturell repräsentierenden Zweck erfüllen. Da außerdem sowohl *Actor* als auch *Component* jeweils eine *Behaviour*-Instanz umfassen, wurden zwei abstrakte Spezialisierungen der Behaviour-Klasse eingeführt (vgl. Abbildung 74): Die Klasse *SimulationBehaviour* stellt die Basis für alle modellierten Verhaltensweisen dar, deren Ausführung anwendungsintern erfolgt. Das *ProxyBehaviour* hingegen realisiert eben jene, für die konzeptionell angedachten Proxy-Funktionalitäten notwendigen Strukturen und Abläufe.

Ein simulativ auszuführendes *SimulationBehaviour* ist, zur weiteren Steigerung der Wiederverwendbarkeit von Teilmodellen als eines der zentralen Ziele dieser Arbeit, unterteilt in mehrere Phasen, die sequenziell durchlaufen werden. Im aktuellen Zustand definiert das Metamodell, grob angelehnt an die vier Phasen motivierten Handelns des Rubikon-Modells [Bak, 2019], die Phasen *Contemplation* (dt.: Reflexion, Betrachtung), *Preparation* (dt.: Vorbereitung), *Action* (dt.: Aktion) und *Maintenance* (dt.: Erhaltung, Verwaltung). Jeder Phase kann wiederum eine beliebige Anzahl an *BehaviourTask*s zugeordnet werden (vgl. Abbildung 75). *BehaviourTask* ist die zentrale Klasse zur Modellierung von dynamischem Verhalten. Der Modellierer kann für konkrete Tasks Subklassen ableiten und die abstrakte Methode *execute* implementieren. Innerhalb eines Tasks ist dabei ohne Weiteres der Zugriff auf den *State*, die *Environment* sowie die dem Wurzelelement zugewiesenen *Goals*, *Actions* und *Preferences* durch Abhängigkeitsinjektion möglich. Ein konkretes Verhalten kann anschließend durch Ableitung

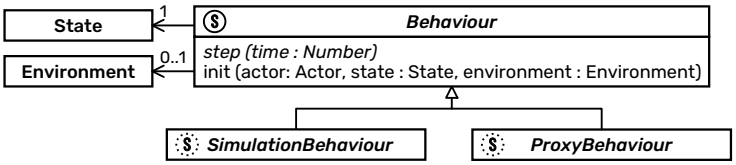


Abbildung 74: Metamodell – Behaviour

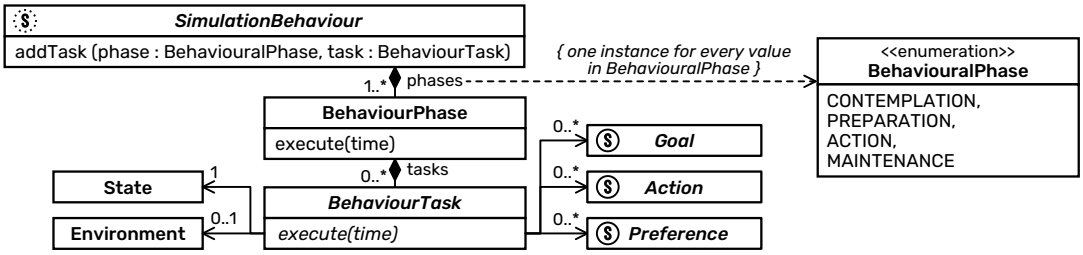


Abbildung 75: Metamodell – *SimulationBehaviour*

der Klasse *SimulationBehaviour* und das Hinzufügen der erstellten Tasks zu den vordefinierten Phasen erstellt werden. Diese Struktur erlaubt die Kopplung von Verhaltensbeschreibungen beliebiger Granularität und die Wiederverwendung bereits existierender konkreter Tasks.

Die *Behaviour*-Variante zur Anbindung externer Testobjekte liegt als konkrete Klasse vor (vgl. Abbildung 76). Semantische Interoperabilität wird dabei dadurch erreicht, dass zu nutzende Werte der Ausgabe des Testobjekts und ihre Eigenschaften vorab definiert und mit internen Werten in Beziehung gesetzt werden. Der genaue Ablauf der anhand dieser Angaben durchgeführten Überführung der Werte vom Fremdmodell in das eigene Zielmodell wird im nachfolgenden Abschnitt „Bidirektionale Informationstransformation“ erläutert. Die Schaffung syntaktischer Interoperabilität hingegen erfordert konkrete Ableitungen der abstrakten Klassen *CommunicationAdapter* und *FormatAdapter*. Ein *ProxyBehaviour* nutzt in einem dreistufigen Verfahren das konfigurierte Dictionary und die zugewiesenen Adapter-Varianten, um mit dem externen Daten auszutauschen. Der genaue Ablauf wird im nachfolgenden Abschnitt „Bidirektionale Vermittlung“ betrachtet.

14.3.2.6 EREIGNISSE, AKTIONEN, ZIELE & WERTE

In Teil II wurde definiert, dass ein Szenario zusätzlich zur statischen Startszene Aspekte enthält, die die Entwicklung der Inhalte der Szene über die Zeit beschreiben. Zu diesen zählen neben den Verhaltensbeschreibungen, welche als Funktion über die Simulationszeit angesehen werden können, diejenigen Szenariobestandteile, die das Verhalten beeinflussen können: Ereignisse, Aktionen, Ziele und Werte (vgl. Abbildung 24).

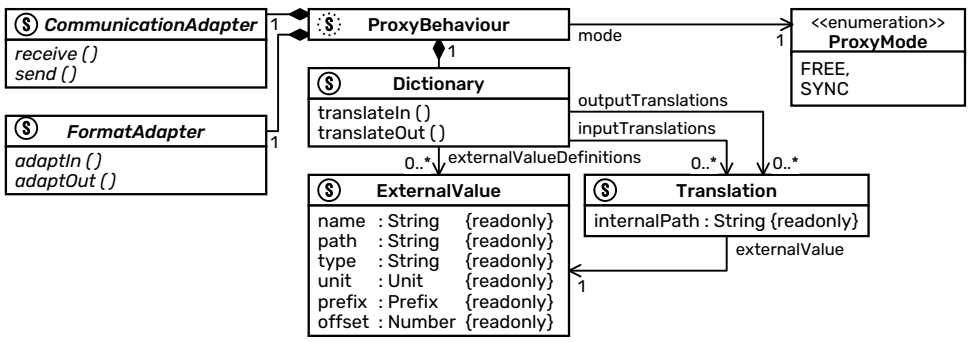


Abbildung 76: Metamodell – *ProxyBehaviour*

Die Metamodellklasse *Event* (vgl. Abbildung 77) bildet Ereignisse ab, indem sie eine einzelne *Property* eines *Elements* eindeutig referenziert und einen Wert (*value*) beinhaltet, der zu einem eindeutig bestimmten Punkt in der Simulationszeit während des Simulationslaufs (*time*) dem *value* Attribut der referenzierten *Property* zugewiesen wird.

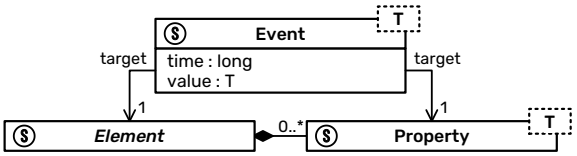


Abbildung 77: Metamodell – *Event* (vgl. *Ereignis* in Abbildung 47)

Die Metamodelklassen *Action*, *Goal* und *Preference* referenzieren ebenfalls ein konkretes *Element*, erheben allerdings zusätzlich die Einschränkung, dass dieses die *Actor*-Schnittstelle implementiert (welche Basisklassen diese Voraussetzung erfüllen, ist Tabelle 14 zu entnehmen). Diese Einschränkung besteht aufgrund des Einflusses auf das Verhalten und der somit gegebenen Notwendigkeit der Existenz eines instanziierten *Behaviours.* Aufgrund der vielfältigen Ausgestaltungsmöglichkeiten von Aktionen, Zielen und Werten sind die entsprechenden Metamodellklassen abstrakter Natur und schreiben keine weiteren strukturellen oder funktionalen Implementierungen vor. Für die Nutzung zur Spezifikation von Szenarioinstanzen müssen somit konkret Subklassen auf den höheren Modellebenen erstellt werden, die Attribute und/oder Methoden enthalten. Anhand der Elementreferenz muss durch die Simulationsanwendung sichergestellt werden, dass die *Action*-, *Goal*- und *Reference*-Objektinstanzen innerhalb der *execute*-Methodenimplementierung aller *BehaviourTasks* des entsprechenden Akteurs verfügbar sind. Zwar kann bei unachtsamer Umsetzung der höheren Modellschichten somit eine hohe Kopplung zwischen konkreten Ausprägungen dieser drei Klassen und konkreten Ausprägungen der Metamodellklasse *SimulationBehaviour* entstehen, aber die Verwendung bleibt zu einem Grad generisch, wie es von einem Metamodell zu erwarten ist. Ein Beispiel der Verwendung von *Goals*, *Actions* und *Preferences* ist im Abschnitt zur *Untersuchung der Eignung des Prototyps* zu finden.

14.3.2.7 EINSCHRÄNKUNGEN

Ähnlich der Ereignisse (*Event*) referenzieren Einschränkungen nach dem in Abschnitt 14.1.2 vorgestellten Grobkonzept ebenfalls Elemente. Im Fall der entsprechenden Klasse *Limitation* wird allerdings zwischen einer einzelnen optionalen Referenz auf die Instanz einer Subklasse von *Element* und einer Menge agierender Elemente, die mindestens eine Referenz umfassen muss, unterschieden. Erste bildet den Verursacher einer Einschränkung ab und zweite die Akteurstypen, die von dieser Einschränkung betroffen sind. Eingeschränkt werden soll eine bestimmte *Property*, genauer gesagt der *value*, dieser. Weil die Art der Einschränkungen eines Wertebereichs sehr vielfältig sein kann, gibt das Metamodell für *Limitations* lediglich vor, dass konkrete Ableitungen die Methode *isValid* implementieren müssen, welche überprüfen soll, ob der übergebene Wert im validen Wertebereich liegt oder nicht. Auch die Menge aller *Limitations*, von denen ein agierendes Element betroffen ist, ist durch Abhängigkeitsinjek-

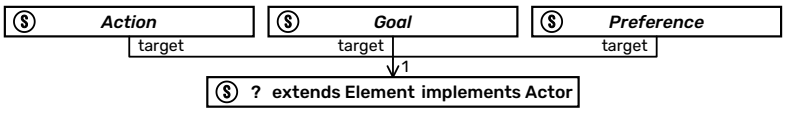


Abbildung 78: Metamodell – *Action*, *Goal* und *Preference* (vgl. *Aktion*, *Ziel* und *Wert* in Abbildung 47)

tion für die Implementierung der *execute*-Methode eines *BehaviourTask* durch die Simulationsanwendung verfügbar zu machen. Ob, in welchem Maße und mit welcher Genauigkeit die definierten Einschränkungen Einfluss auf das jeweilige Verhalten haben, ist dabei der Verhaltensimplementierung überlassen und kann z. B. durch die für das betroffene Element spezifizierten Wertevorstellungen in Form der *Preferences* beeinflusst werden.

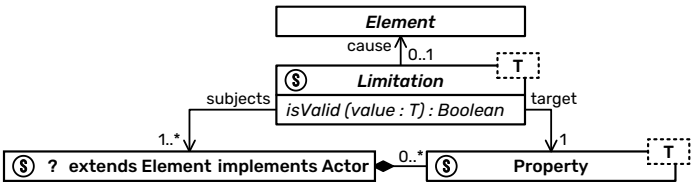


Abbildung 79: Metamodell – *Limitation* (vgl. *Einschränkung* in Abbildung 47)

14.3.2.8 SZENARIO

Der letzte zentrale, aber nicht weniger relevante, Bestandteil des Metamodells ist die konkrete Klasse *Scenario* (vgl. Abbildung 80). Objektinstanzen dieser Klasse fungieren als zentrale Einheit zur Bündelung aller zuvor beschriebenen Inhalte eines konkreten, simulativ ausführbaren Szenarios. Die Zusammensetzungsstrukturen sind dabei analog zu der Definition in Abschnitt 7.2.4 bzw. der konzeptuellen Erweiterung dieser in Abschnitt 14.1.2 modelliert.

Zusätzlich umfasst jede Objektinstanz der Klasse *Scenario* ein *Configuration*-Objekt, welches alle Angaben enthält, die eher technischen Ursprungs sind, für eine simulative Durchführung zu Testzwecken aber notwendig sind. So ist in Form eines *Library*-Objektes dort die Information verortet, wo die Modellbibliothek zu finden ist, auf welche sich die Szenariospezifikation beruft. Diese Angabe ist unerlässlich für die simulative Ausführung, da aufgrund der Anforderung, dass Szenarioinstanzen persistierbar und austauschbar sein müssen, diese in serialisierter textueller Form vorliegen müssen. Um aus diesen das initiale Laufzeitmodell zu initialisieren, muss die Simulationsanwendung entsprechend des Konzeptes Zugriff auf die entsprechende Modellbibliothek haben.

Die Einordnung von *Elementen* der Startszene geschieht abweichend von der konzeptionellen Abbildung 47 in den zwei voneinander getrennten Mengen *scenery* und *actors*. Dadurch wird die Übersichtlichkeit der Struktur verbessert. Auch ist die eigenständige Betrachtung aller passiven physischen Elemente eines Szenarios oder Simulationslaufs unter dem Begriff Szenerie bereits in einigen anderen Arbeiten, wie [Geyer et al., 2014], anzutreffen und wurde für sinnvoll erachtet.

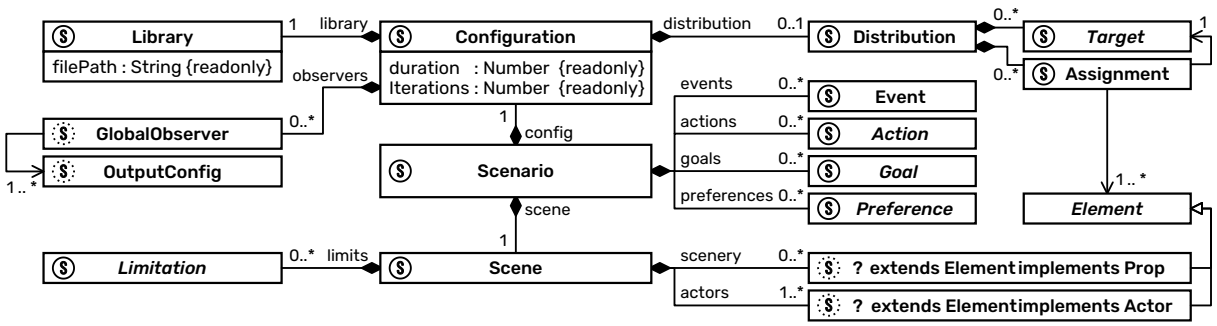


Abbildung 80: Metamodell – *Scenario* als Zusammenfassung der Inhalte einer Szenarioinstanz

14.3.3 SIMULATION

Auf Basis des Metamodells können durch mehrstufige Vererbungen im Sinne des in Abschnitt 14.1.3 vorgestellten Konzeptes konkrete Modelle erstellt werden. Zusammengefasst in Form einer Szenariobibliothek im Sinne des Konzeptes aus Abschnitt 14.1.5.3 können die Modellbestandteile anschließend als Basis zur Spezifikation von Szenarioinstanzen genutzt werden. Eine solche wird dabei in einer persistierbaren serialisierten Repräsentation der hierarchischen Modellinstanzen (vgl. Abbildung 80) erstellt. Für diesen Zweck ist die Verwendung textbasierter Serialisierungsformate wie der *Extensible Markup Language* (XML) oder der *JavaScript Object Notation* (JSON) naheliegend, da diese auch durch den Menschen direkt lesbar und erstellbar sind. Zentrale Aspekte der Überführung einer solchen Repräsentation einer Szenarioinstanz in einen konkreten HLA-konformen Simulationslauf sowie zentrale funktionale Aspekte des Simulationsablaufs werden nachfolgend genauer beschrieben.

Auch im Rahmen der nachfolgenden Betrachtung der inhaltlichen Ausgestaltung der Konzepte sei angemerkt, dass lediglich die relevantesten Konzeptteile im Detail dargestellt werden. Die an dieser Stelle ausgelassenen Abläufe und Strukturen lassen sich aber aus dem öffentlich zugänglichen Programmcode der Umsetzung erschließen. Die experimentelle Umsetzung ist Thema des Abschnitts 15.

14.3.3.1 MODELLÜBERFÜHRUNG

Die Simulationsanwendung muss in der Lage sein, eine, in einem wie einleitend beschriebenen Format vorliegende, Szenarioinstanz entgegenzunehmen und, in Kombination mit der referenzierten Modellbibliothek, die zur Ausführung notwendigen Objekte und Strukturen zu initialisieren (vgl. Anforderung LA08 und Abbildung 63). Die Simulationsanwendung extrahiert daher zunächst die Information bezüglich der Modellbibliothek aus der gegebenen, serialisiert vorliegenden Szenarioinstanz. Diese wird anschließend geladen und eine Teilmenge der enthaltenen Klassen wird entsprechend den Angaben in der vorliegenden serialisierten Repräsentation der Szenarioinstanz initialisiert (vgl. Abbildung 81). Dieses Vorgehen entspricht somit dem Vorgehen des *Unmarshalling*, wie es u. a. im RFC 2713 in Bezug auf die objektorientierte Programmiersprache *Java* beschrieben wird: „To ‚marshal' an object means to record its state and codebase(s) in such a way that when the marshalled object is ‚unmarshalled', a copy of the original object is obtained, possibly by automatically loading the class definitions of the object." [Ryan et al., 1999]

Die entstandenen Elementinstanzen werden anschließend einem *Federate* Objekt zugewiesen, welches als Ausführungscontainer dient (vgl. Abbildung 81). Alle drei möglichen Federate-Typen haben ge-

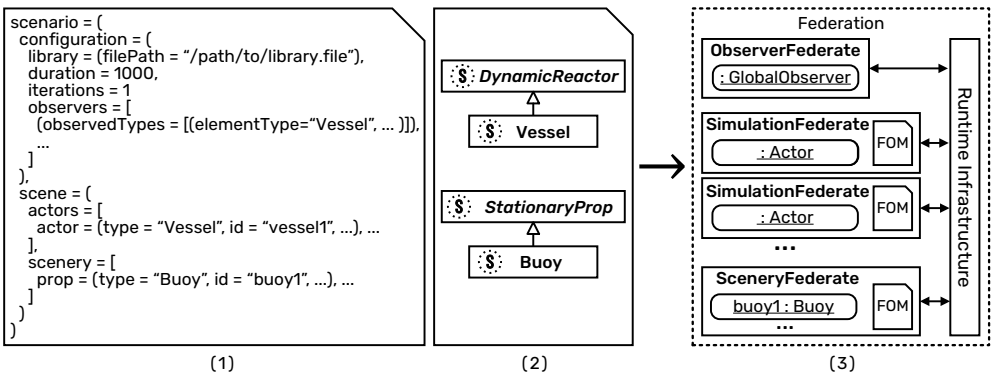


Abbildung 81: Abstrahierte Veranschaulichung des Unmarshallings einer Szenarioinstanz (1) unter Nutzung der referenzierten Modellbibliothek (2) inklusive der Überführung in HLA-konforme Federates (3).

meinsam, dass sie die gesamte administrative Kommunikation mit der RTI vor und während des Simulationslaufs durchführen. Sie unterscheiden sich hauptsächlich in drei Punkten: (1) der Menge an *Elementen,* für die sie zuständig sind, (2) den Funktionalitäten, die sie bezüglich des Veröffentlichen und Abonnierens von Objekten und Attributen unterstützen, und (3) der Ausführung von möglichen Verhaltensbeschreibungen der zugewiesenen *Elemente.* Nachfolgend sind die drei Federate-Typen bezüglich ihrer Ausprägung dieser drei Charakteristiken kurz beschrieben:

SimulationFederate: Für jedes *Element*, das die *Actor* Schnittstelle direkt oder indirekt implementiert, d. h. ein auszuführendes *Behaviour* besitzt, wird ein eigenes *SimulationFederate* instanziiert. Dieser Federate-Typ verwaltet entsprechend der Konfiguration des *Elements* alle eingehenden und ausgehenden Datenflüsse und führt diese durch. Bei Fortschreiten der lokalen Simulationszeit werden alle Verhaltensbeschreibungen des Elements und seiner Komponenten ausgeführt.

SceneryFederate: Alle *Elemente*, die die *Prop* Schnittstelle direkt oder indirekt implementieren, werden einem gemeinsamen SceneryFederate als Ausführungscontainer zugewiesen. Dieser Federate-Typ verwaltet entsprechend der Konfiguration der Elemente die ausgehenden Datenflüsse und führt diese einmalig zu Beginn des Simulationslaufs durch. Es werden keine Updates bei der RTI abonniert und bei Fortschreiten der Simulationszeit wird kein Verhalten durchgeführt.

ObserverFederate: Für jedes *GlobalObserver* Element, das in der *Configuration* eines Szenarios spezifiziert ist, wird ein ObserverFederate erzeugt. Dieser Federate-Typ registriert entsprechend der Konfiguration Abonnements bei der RTI. Ausgehend werden keine eigenen Registrierungen durchgeführt oder Aktualisierungen gesendet. Bei Fortschreiten der Simulationszeit wird der aktuelle Stand der abonnierten Daten entsprechend der Konfiguration ausgegeben.

14.3.3.2 MODELLTRANSFORMATION

Wie im Rahmen der Technologieauswahl in Abschnitt 14.3.1 bereits beschrieben wurde, erfordert HLA die OMT-konforme Repräsentation der Daten, die zwischen den am Simulationslauf teilnehmenden Federates ausgetauscht werden können. Somit müssen entweder die Inhalte eines Szenarios zusätzlich in eine OMT konforme Repräsentation überführt werden oder einen Schritt vorher alle Inhalte einer Modellbibliothek. Für die hier angedachten, in ihrer konkreten Ausprägung höchstvariablen, Modellbibliotheken würde dies hohen manuellen Aufwand bedeuten. Da zusätzlich die Anforderungen LA15–17 eine manuelle Überführung der Modelle verbieten, müssen die entsprechenden OMT-konformen Repräsentationen generativ erzeugt werden (vgl. [Möller & Antelius, 2010]).

Eine Szenarioinstanz beschreibt vollständig, welche Daten in einer Federation zur Simulationslaufzeit vorhanden sein können. Zusätzlich sind aufgrund der zentralen Orchestrierung der Federation keine Veränderungen im Datenmodell durch das Ein- oder Austreten weiterer Federates zur Laufzeit zu erwarten. Das FOM als zentrales Datenmodell einer Federation kann aus einer Menge von FOM-Modulen durch die RTI zusammengeführt werden. In Hinblick auf die zuvor beschriebene Aufteilung der Modellelemente auf mehrere eigenständige Ausführungscontainer unterschiedlichen Funktionsumfangs ist es daher naheliegend, dass für jeden dieser Container ein individuelles FOM-Modul erzeugt wird, das die diesem Container zugewiesenen Elemente repräsentiert (vgl. Abbildung 81).

Während der Initialisierung eines Simulationslaufs wird somit pro Federate-Container ein FOM in Form einer OMT-konformen XML-Datei erzeugt. Zu diesem Zweck wird die Vererbungshierarchie al-

Listing 1: OMT-konforme XML-Repräsentation der Modellelemente (gekürzt)

```
<objects>
   <objectClass>
      <name>HLAobjectRoot</name>
      <objectClass>
         <name>Element</name>
         <sharing>Publish</sharing>
         <objectClass>
            <name>PhysicalElement</name>
            <sharing>Publish</sharing>
            ...
```

Listing 2: OMT-konforme XML-Repräsentation einer *Property* durch jeweils drei Attribute (gekürzt)

```
<objectClass>
   <name>DynamicPhysicalElement</name>
   <sharing>Publish</sharing>
   <attribute>
      <name>rotation</name>
      <dataType>HLAfloat64BE</dataType>
      <sharing>Publish</sharing>
   </attribute>
   <attribute>
      <name>rotation.unit</name>
      <dataType>HLAASCIIstring</dataType>
      <sharing>Publish</sharing>
   </attribute>
   <attribute>
      <name>rotation.unit.prefix</name>
      <dataType>HLAASCIIstring</dataType>
      <sharing>Publish</sharing>
   </attribute>
<objectClass>
```

ler diesem Container zugewiesenen *Element*-Instanzen durchlaufen und in eine Struktur geschachtelter *objectClass* XML-Elemente überführt (vgl. Listing 1). Der angegebene *name* entspricht dabei dem jeweiligen Klassenbezeichner. Das Sub-Element *sharing* wird mit dem Inhalt „Publish“ versehen, wenn die Klasse mindestens ein Feld des Datentyps *Property* besitzt und mindestens eine dieser *Property*-Instanzen für das Feld *publish* den Boolean-Wert *true* hält. Ansonsten wird „Neither“ zugewiesen. Die *objectClass*-Elemente werden anschließend entsprechend des OMT mit *attribute*-Elementen befüllt. Zur Abbildung der für Umrechnungen nötigen Angaben der Unit und des Präfixes werden für jede *Property* der übergeordneten Klasse zwei zusätzliche *attribute*-Elemente erzeugt (vgl. Listing 2).

14.3.3.3 IMPLIZITE INFRASTRUKTURANBINDUNG

Die Entscheidung für die Gestaltung der Simulationsanwendung entsprechend des HLA-Standards bringt zusätzliche Komplexität mit sich. Diese Tatsache steht im Gegensatz zum Teilziel 3 (Z3) und der diesem zugeordneten lösungsbezogenen Anforderung 16 (LA16). In erster Konsequenz wurde die Entscheidung für HLA bereits unter der Voraussetzung getroffen, dass diese zusätzliche Komplexität so weit gekapselt werden muss, dass weder der Modellentwickler noch der Tester in direkten Kontakt zu dieser kommen. Als erste grobe Annäherung wurde das Konzept des *Agentengerüsts* vorgestellt.

Bereits im Jahr 1999 haben Schulze et al. mögliche Ansätze diskutiert, um die Entwicklung HLA-konformer verteilter Simulationsanwendungen zu vereinfachen [Schulze et al., 1999]. Dabei nennen sie auch die potenzielle Möglichkeit, die HLA-Funktionalitäten vollständig vor dem Modellentwickler zu verstecken. Da sämtliche Kommunikation mit der RTI implizit geschieht, verwenden sie die Bezeichnung des *impliziten Lösungsansatzes*. Schulze et al. nennen vier Federate-spezifische Aufgaben, die es als Mindestvoraussetzung zu automatisieren gilt, um ein solches vollständiges Verstecken zu ermöglichen:

(1) Synchronisation mit der RTI und den anderen teilnehmenden Federates
(2) Konvertierung zwischen modell-/anwendungsspezifischen und OMT-konformen Datentypen, wie sie im FOM definiert sind und für die Kommunikation mit der RTI genutzt werden
(3) Automatische Überführung von Änderungen der lokalen Modellvariablen in HLA-Object- und Attribute-Updates sowie die entsprechende Kommunikation dieser
(4) Automatische Instanziierung lokaler Kopien von veröffentlichten Objekteinstanzen der anderen teilnehmenden Federates sowie Zuordnung von HLA-Attribute-Updates zu entsprechenden Modellvariablen

Um die HLA-Funktionalitäten möglichst vollständig zu verbergen, ist es daher naheliegend, die zwei vermittelnden Schnittstellenimplementierungen *RTI Ambassador* und *Federate Ambassador, die* durch den HLA-Standard bereits vorgeschrieben sind, gemäß des *Adapter* Entwurfsmusters in einem einzelnen *RTI-Adapter* zusammenzufassen, um die genannten Funktionalitäten zu erweitern und eine Schnittstelle zur Nutzung durch, dem zuvor vorgestellten Metamodell entsprechende, Elemente anzubieten (vgl. Abbildung 82).

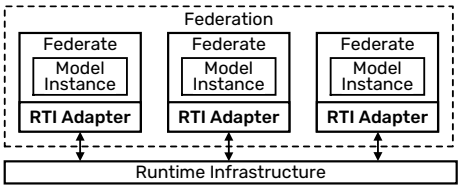


Abbildung 82: Integration eines Adapters als Vermittlerschicht zur Kapselung und Automatisierung der Kommunikation der *Federates* mit der zentralen *Runtime Infrastructure (RTI)*

Federate-Instanzen stellen hier Laufzeitcontainer für Modellelemente dar. Sie übernehmen sowohl die Steuerung dieser (vgl. Beschreibung der Federate-Typen in Abschnitt 14.3.3.1) als auch administrative Aufgaben, wie den Beitritt zur Federation. Der RTI-Adapter muss somit zwei Arten von Schnittstellen zur Verfügung stellen: eine zur Kommunikation des Federate-Containers mit der RTI bezüglich Verwaltungs- und Synchronisationsaufgaben (1) und eine, anhand welcher während der Verhaltensausführung auf stets aktuelle abonnierte Daten zugegriffen werden kann und Veränderungen am eigenen Zustand automatisiert in der Federation veröffentlicht werden (2–4). Die vier zu automatisierenden Aufgaben, die von Schulze et al. genannt wurden, werden dementsprechend durch eine Menge zusammenwirkender Komponenten übernommen. Diese werden durch den *Federate*-Laufzeitcontainer instanziiert und koordiniert. Diese zusätzliche Abstraktionsschicht ermöglicht einen modellbezogenen kommunikativen Austausch zwischen den Modellelementen über die RTI, und das Konzept bzw. die Gesamtheit der Bestandteile der adaptierenden Schicht wird im weiteren Verlauf als *Model-Aware RTI-Exchange (maRTIx)* bezeichnet. Der entsprechende Aufbau sowie Bezugnahmen und Kontroll- und Kommunikationsflüsse sind in Abbildung 83 dargestellt.

Die Automatisierung von Aufgabe (2) erfordert, besonders in Hinblick auf zusammengesetzte Datentypen wie *Position*, Wissen darüber, welche FOM-Objektklassen und FOM-Attribute welche *Element*-Instanzen und welche *Properties* des Modells repräsentieren. HLA arbeitet zudem mit sogenannten *Handles,* welche zur eindeutigen Identifikation veröffentlichter Objektinstanzen und Attribute genutzt werden. Um *Handles* zu erzeugen, wird der Name bzw. Pfad des Objektes bzw. des Attributs im FOM benötigt. Zur Simulationslaufzeit existieren somit vier verschiedene Repräsentationen inhaltlich nahe-

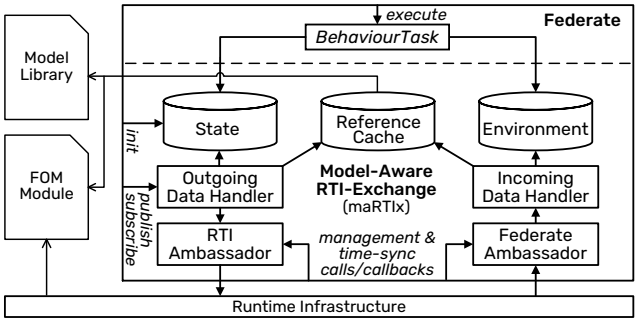


Abbildung 83: Detailentwurf des RTI-Adapter zur Kapselung und Automatisierung der Kommunikation

zu deckungsgleicher Konstrukte: (a) die Klasse als Bestandteil der aktuellen Modellbibliothek, (b) eine Referenz der genutzten Programmiersprache auf eine Instanz dieser Klasse, (c) das entsprechende XML-Element des FOM-Moduls und (d) das von der RTI ausgegebene *Handle*-Objekt. Um ein Mapping im Sinne von (2) zu ermöglichen, werden im zentralen *ReferenceCache* des Adapters alle während des gesamten Lebenszyklus des Federates erzeugten Repräsentationen aller vier Arten flüchtig gespeichert. Dabei werden zusätzlich die Beziehungen zwischen den genannten Repräsentationsarten und zusätzlich die eindeutigen *id*-Werte der Modellinstanzen in einer Art und Weise gehalten, die eine Navigation von einer der Repräsentationen/Referenzen zu einer beliebigen anderen ermöglicht. Der *ReferenceCache* erlaubt es somit, für eine *Property*-Instanz direkt den passenden FOM-Pfad und ein eventuell vorhandenes HLA-Laufzeit-Handle zu bekommen. Anhand dieser kann der *Outgoing Data Handler* (vgl. Abbildung 83) den HLA-seitig genutzten Datentyp identifizieren, den *value* der Property entsprechend in diesen überführen und anhand des korrekten *AttributeHandle* die RTI über eine Aktualisierung des Attributes benachrichtigen. Eingehende Daten werden vom *Incoming Data Handler* größtenteils analog zu dem beschriebenen Vorgehen behandelt.

14.3.3.4 ZENTRALE ZUSTANDSHALTUNG

Der Datenaustausch zwischen Teilsystemen und Systemkomponenten eines Fahrzeugs erfolgt in den meisten Fällen über Bussysteme wie den *Controller Area Network* (CAN)-Bus und auf entsprechenden Systemen aufsetzende Netzwerk- und Kommunikationsprotokolle wie NMEA 2000 und NMEA 0183 der *National Marine Electronics Association* (NMEA) [Borgeest, 2023; Kessler, 2021]. Eine solche Kommunikationsstruktur sollte daher auch durch die Modelle abzubilden sein und während des Simulationslaufs von den Komponenten genutzt werden können, um Daten untereinander auszutauschen. Eine erste Annäherung wird durch die Einführung der bereits im Rahmen der Beschreibung des Metamodells erwähnten zentralen Zustandshaltung in Anlehnung an die gängigen Definitionen eines agentenorientierten Aufbaus erreicht, dessen Aufbau in Abbildung 84 gekürzt dargestellt ist.

Der interne Zustand wird zentral durch eine Instanz der Klasse *State* abgebildet. Ein *State* umfasst eine beliebige Menge Instanzen der Klasse *StateEntry*, welche ausschließlich über die öffentlichen Methoden des *State*-Objekts verwaltet werden können. Ein *StateEntry* bildet dabei zu einem Großteil diesel-

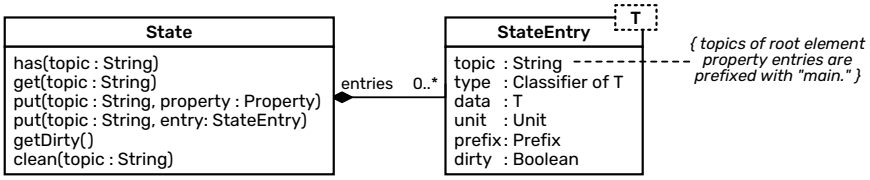


Abbildung 84: Zentrale Zustandshaltung einer *SimulationFederate*-Instanz

ben Informationen ab wie die Klasse *Property*. In Anlehnung an die u. a. im Internet-of-Things Bereich häufig anzutreffenden *Message Bus*-Architekturen und entsprechende Protokolle wie MQTT[77] und RabbitMQ[78] muss jedem *StateEntry* ein Thema (engl.: topic) zugewiesen werden. So kann sichergestellt werden, dass ein *StateEntry* bei der Inter-Komponenten-Kommunikation eindeutig zugeordnet und gezielt genutzt werden kann. Für einen *topic*-String ist syntaktisch ein einzelner Punkt zur Aneinanderreihung von Unterthemen vorgesehen. Zu Beginn des Lebenszyklus eines *SimulationFederates* wird für jede *Property* des zugewiesenen *Actors* ein *StateEntry* erzeugt. Dies gilt auch für *Properties* tieferer Ebenen komplexer Datentypen und die *Property*-Felder aller Superklassen. Das *topic* eines solchen initial erzeugten Eintrags setzt sich zusammen aus dem Präfix „main“ und dem entsprechenden Namen der *Property*-haltenden Felder, separiert durch einen Punkt (Beispiel: *main.position.x*).

Wird der Wert *data* einer der *StateEntrys* dieser zustandsbildenden Größen während der Ausführung der Verhaltensmodelle aktualisiert, dann wird dieser *StateEntry* als *dirty* markiert. Das Fortschreiten in der Simulationszeit abschließend, übernimmt das Federate diese aktualisierten Wertezuweisungen für die entsprechenden *Property*-Instanzen. Gleichzeitig wird der *Outgoing Data Handler* über das Ende des Zeitschritts informiert, welcher entsprechende *AttributeUpdates* unter Verwendung des *Reference Stores* erzeugt und mithilfe des *RTI-Ambassadors* an die RTI kommuniziert.

14.3.3.5 MODELLINFERENZ

Der HLA-Standard bietet zwar einen standardisierten Rahmen für Objektmodelle, stimmt aber nicht vollständig mit den typischen, wie man sie aus dem Bereich der objektorientierten Analyse- und Entwurfsmethodiken oder dem objektorientierten Programmierparadigma kennt, überein [Tolk, 2001]. Die von Schulze et al. als *Ghosting* bezeichnete automatische Instanziierung lokaler Kopien von über die RTI veröffentlichten Objekten erfordert jedoch eine Abbildung empfangener OMT-konformer Daten auf die objektorientierte Struktur der geladenen Modellbibliothek.

Verschiedene Veröffentlichungen haben bereits das Konzept einer objektorientierten Nutzung der HLA aus unterschiedlichen Richtungen betrachtet. Möller & Antelius [2010] beschreiben Gemeinsamkeiten und Unterschiede objektorientierter und OMT-konformer Datenmodelle und zeigen auf einer abstrahierten Betrachtungsebene, wie eine mögliche objektorientierte Middleware für HLA-basierte Simulationsanwendungen aussehen könnte. Abschließend identifizieren sie einige Herausforderungen und potenzielle Probleme, die der Versuch einer Abbildung objektorientierter Daten auf das OMT mit sich bringen kann. In dem in dieser Arbeit konkret betrachteten Anwendungsfall sind die meisten dieser aber nicht oder nur zu einem geringen Teil zutreffend.

Kirov [2015] stellte eine Architektur vor, die den Austausch von Informationen mit Hilfe von objektorientierten Instanzen kapselt. Dabei war das vorrangige Ziel die Verschleierung der Kommunikationskomplexität und nicht das objektorientierte Vorgehen selbst. Im Rahmen des vorgestellten Architekturkonzepts hat jedes Objekt zusätzlich zur Datenhaltung Methoden zur Kommunikation von HLA-*AttributeUpdates* auf Senderseite und zum Empfangen und Verarbeiten von solchen auf Empfängerseite. Diese Struktur geht konträr zu dem fundamentalen Designprinzip der *Separation of Concerns* (SoC) (dt.: Trennung von Zuständigkeiten), verhindert klar abgrenzbare, saubere Programmstrukturen und gilt im Allgemeinen als nicht erstrebenswert. Überdies wird die Verantwortung

[77] OASIS Open, *MQTT - The Standard for IoT Messaging*: `https://mqtt.org/` [letzter Zugriff: 17.02.26]
[78] Broadcom Inc., *RabbitMQ: One broker to queue them all / RabbitMQ*: `https://www.rabbitmq.com/` [letzter Zugriff: 17.02.26]

für das Erzeugen sowie Empfangen und Umsetzen von Aktualisierungen so auf den Entwickler der Modellklasse übertragen, was den hier formulierten Zielen und Anforderungen klar widerspricht.

Diese Arbeit hat zum Ziel, ein objektorientiertes Umgebungsmodell des Federates als Agent aus der Menge aller empfangenen Daten herzuleiten. Die verteilte Ausführung, die Abstraktion und die Überführung in OMT-Datenstrukturen stellen Herausforderungen für eine solche Modellinferenz dar, welche einen gemeinsamen formalen Rahmen der teilnehmenden Federates erfordern. Dieser ist durch das, im vorliegenden Konzept allen Modellbibliotheken zugrunde liegende, einheitliche Metamodell gegeben. In Verbindung mit dem RTI-Adapter und dessen zentralem *Reference Store* lassen sich die von anderen Federates verwalteten Element-Instanzen aus den von der RTI empfangenen OMT-Daten rekonstruieren. Abbildung 85 veranschaulicht das beschriebene Prinzip: Das Federate *Teil-Simulation 1* veröffentlicht Aktualisierungen (*akt*) zu dem von ihm verwalteten Element *Akteur 1*. Die RTI als Kommunikationsschnittstelle liefert die entsprechenden OMT-Daten an das Federate *Teil-Simulation 2*, welches eine Instanz derselben Klasse als lokale Kopie der Elementinstanz *Akteur 1* erzeugt und die Aktualisierungen auf diese überträgt.

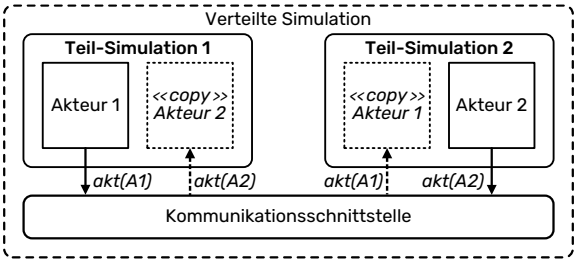


Abbildung 85: Ghosting von Objekten anderer Simulationsteilnehmer

Derart inferierte Kopien werden von der, innerhalb jedes Federates einzigartigen, *Environment*-Instanz gehalten, welche als zentrale Schnittstelle für die Verhaltensimplementierungen zum Zugriff auf das Umgebungsmodell dient. Wie in Abbildung 86 angedeutet, bietet die konkrete Klasse Environment zu diesem Zweck verschiedene öffentliche Methoden zum Abfragen von Element-Instanzen.

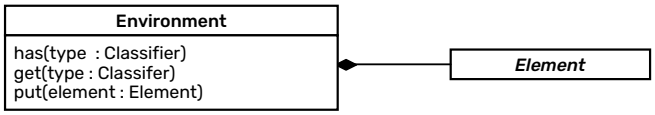


Abbildung 86: Ghosting von Objekten anderer Simulationsteilnehmer innerhalb einer kooperativen Simulation

14.3.3.6 KOMPONENTENBASIERTE WIRKSTRUKTUREN

Der konzipierende Abschnitt 14.1.1 hat die Bedingung eingeführt, dass sowohl eine serielle als auch eine parallele Ausführung der komponentenbasierten Wirkstruktur eines Akteurs möglich sein muss. Die Möglichkeit zur Definition der Ausführungsreihenfolge individueller Komponenten durch das Feld *executionGroup* der abstrakten Klasse *Component* (vgl. Abschnitt 14.3.2.2) legt den Grundstein zur Erfüllung dieser Bedingung. In Kombination mit der Möglichkeit, einem Akteur im Rahmen der Szenariospezifikation beliebig breite und tiefe Komponentenhierarchien zuzuweisen, werden so flexible komplexe Wirkstrukturen ermöglicht.

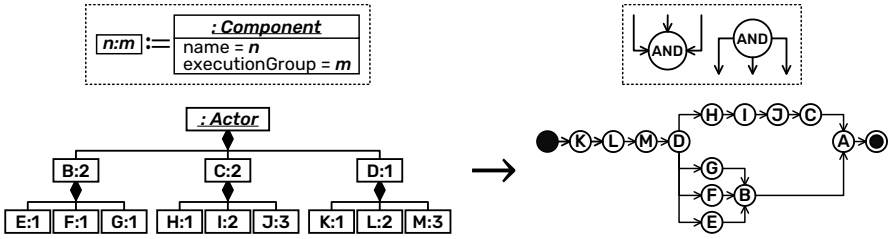


Abbildung 87: Komponentenhierarchie eines Akteurs (links) und der entsprechende Verhaltensgraph (rechts)

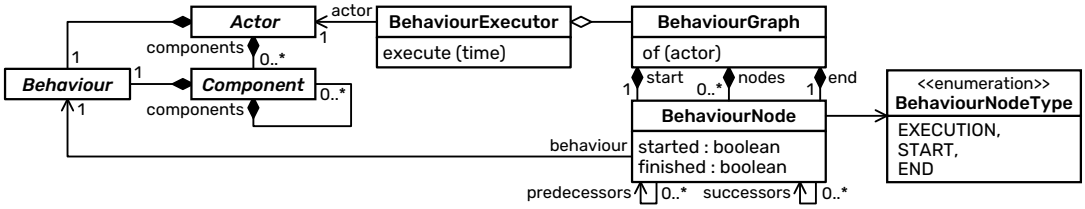


Abbildung 88: Struktur und Beziehungen des *BehaviourGraph* eines *Actors*

Komponentenhierarchien haben stets eine Baumstruktur (vgl. Abbildung 87). Um den Aufwand während der Simulationsschritte selbst zu minimieren, werden die *Behaviour*-Instanzen der Knoten des Baumes entsprechend ihrer Position und ihrer zugewiesenen Ausführungspriorität in einen gerichteten, zyklenfreien Graphen mit genau einem Start- und einem Endknoten überführt (vgl. Abbildung 87). Das einmalige Durchlaufen dieses *BehaviourGraph* (vgl. Abbildung 88) wird bei jedem Voranschreiten in der Simulationszeit durch den Federate-Container angestoßen. Dabei wird ausgehend vom Startknoten wiederholt die Ausführung der Verhaltensimplementierung des Nachfolgerknotens angestoßen und nach Abschluss der Durchführung zu diesem navigiert, bis der Endknoten erreicht wurde. Hat ein Knoten mehrere Nachfolger, so wird der beschriebene Ablauf in einem eigenen Thread pro Nachfolgerknoten angestoßen. Wird ein Knoten erreicht, der mehrere Vorgängerknoten referenziert, wird entsprechend gewartet, bis diese alle als abgeschlossen markiert wurden, bevor mit der Traversierung des Graphen fortgefahren wird.

Die Struktur des *BehaviourGraph*, welche nach einmaliger Initialisierung keine weiteren Änderungen erfährt, und das eindeutig geregelte Traversieren von diesem stellen die stets korrekte Abfolge der Verwendung und Aktualisierung des internen Zustands entsprechend der Konfiguration durch die Szenarioinstanz sicher. Eine solche temporale Kausalität der Abläufe ist ausschlaggebend für hochwertige Simulationsergebnisse [Bononi et al., 2005].

Die Flexibilität des Einsatzes möglicher Verhaltensimplementierungen wird weiter gesteigert durch die Möglichkeit, die Schrittweite von Akteuren und Komponenten als Subklassen der *Steppable* Schnittstelle (vgl. Abbildung 68) individuell zu spezifizieren. Zwar hat eine Schrittweite, die kleiner ist als die der Gesamtsimulation, keine Auswirkung auf den Ablauf, da die Verhaltensausführungen zentral vom Federate-Container ausgelöst werden. Eine Schrittweite größer als die der Gesamtsimulation ermöglicht es einzelnen Komponenten, bestimmte Zeitschritte zu überspringen: Beträgt die konfigurierte globale Simulationsschrittweite zum Beispiel 1 und die Schrittweite einer Komponente 3, dann wird das *Behaviour* dieser Komponente nur jeden dritten Simulationsschritt ausgeführt.

Trotz des Fortschreitens der Gesamtsimulation mit einer festen Schrittweite sind ebenfalls kontinuierlich Verhaltensimplementierungen möglich, deren Ausführung unabhängig vom Fortschreiten des Federate-Containers in der Simulationszeit ist: Eine konkrete Implementierung von *BehaviourTasks* kann

für etwaige Berechnungen eigene Threads erstellen, welche parallel zum zyklischen Ablauf des Federate-Containers ausgeführt werden. Ein solches Vorgehen geht aber zulasten der zuvor beschriebenen kausalen Ordnung und kann daher zu nichtdeterministischen Abläufen führen.

14.3.4 VERTEILTE AUSFÜHRUNG

Spätestens das Konzept zur Anbindung externer Testobjekte erfordert eine mindestens teilweise physische Verteilung der Simulationsteilnehmer. Die zentrale Herausforderung dabei ist die Sicherstellung eines temporal deterministischen Ablaufs der Kommunikation. Das heißt, dass zwischen den Teilnehmern übertragene Nachrichten in der korrekten Reihenfolge der Simulationszeit entsprechend verarbeitet werden müssen, unabhängig von den tatsächlichen Ausprägungen der verwendeten Kommunikationswege (vgl. [Gütlein, 2023]). Tolk [2012b] definiert etwas feingranularer drei Anforderungen, die es durch eine verteilte Simulationsanwendung zu erfüllen gilt. Diese werden durch den Einsatz von HLA erfüllt und bedürfen an dieser Stelle keiner weiteren Betrachtung, sollen aber der Vollständigkeit halber trotzdem genannt werden:

Effektivität: Alle kommunizierten Informationen müssen an die korrekten (interessierten) Simulationsteilnehmer geliefert werden.

Effizienz: Ausschließlich die von dem spezifischen Simulationsteilnehmer benötigten Informationen dürfen diesem geliefert werden.

Korrektheit: Die Auslieferung muss zur korrekten Zeit geschehen (temporale Kausalität) oder für den Simulationsteilnehmer muss die korrekte Reihenfolge ersichtlich sein.

Drei zentrale Aspekte der entworfenen Lösung bzgl. der verteilten Ausführung, die nicht bereits implizit betrachtet wurden und nicht direkt durch HLA abgedeckt sind, werden nachfolgend beschrieben.

14.3.4.1 BIDIREKTIONALE VERMITTLUNG

Syntaktische und semantische Interoperabilität sind nötig, um sinnvoll mit dem externen Testobjekt zu kommunizieren. Das übergreifende Lösungskonzept zur Erfüllung dieser Anforderung wurde in den Abschnitten 14.1.6 und 14.1.7 beschrieben und mit entsprechenden Metamodell-Strukturen in Abschnitt 14.3.2.5 angereichert.

Um die gewünschte Interoperabilität zu erreichen, ist eine bidirektionale Überführung von Daten und Informationen notwendig (vgl. u. a. [Tibba et al., 2016]). Statt die Umsetzung einer allumfassenden Transformationsfunktion vorauszusetzen, wurden im Sinne des zentralen Ziels der Modularität und Wiederverwendbarkeit drei Transformationsebenen identifiziert:

Kommunikationsprotokoll: Das verwendete Kommunikationsprotokoll regelt den Austausch zwischen der Simulationsanwendung und dem externen Testobjekt. Neben gängigen Netzwerkprotokollen wie dem *User Datagram Protocol* (UDP) und dem *Transmission Control Protocol* (TCP) sind auf dieser Ebene auch andere, weniger standardisierte Vorgehen denkbar, wie der Austausch über das abwechselnde Lesen und Schreiben einer Datei, auf die per *File Transfer Protocol* (FTP) zugegriffen wird.

Datenaustauschformat: Können Daten mit dem externen Testobjekt technisch ausgetauscht werden, muss auf der nächsten Ebene das vom Testobjekt verwendete Format zur Strukturierung der kommunizierten Daten verstanden werden können. Ebenso müssen vom Testobjekt empfangene Daten in ein Format, das von der nächsten Schicht interpretier- und nutzbar ist, umgewandelt werden. Bei ausgehender Kommunikation müssen analog dazu, dem internen Format entsprechende, Daten in das vom Testobjekt erwartete überführt werden. Auf dieser Ebene kann z. B. eine Überführung von textbasier-

ten *JavaScript Object Notation* (JSON)-Inhalten oder einzelnen standardisierten NMEA 0183-Nachrichten in das interne Format notwendig sein.

Informationsmodell: Auf dieser Ebene wird die Bedeutung der übertragenen Daten betrachtet. Dabei sind sowohl die Struktur der Daten als auch die verwendeten Datentypen und eventuell vorhandene Einheiten und Präfixe einzelner Werte innerhalb der Struktur relevant. So könnte ein externes Testobjekt etwa die Geschwindigkeit eines Schiffes in *m/s* in einer geschachtelten Struktur liefern, in welcher sich innerhalb von *VslInfo* das Feld *spd* befindet. Die Geschwindigkeit eines internen Schiffsmodells könnte gleichzeitig als *Property*-Feld *speed* einer von *Element* und *Akteur* abgeleiteten Klasse *Ship* modelliert und mit der Einheit *knots* versehen sein. Dann muss auf dieser Transformationsebene bei eingehender Kommunikation der Wert mit dem Pfad *VslInfo.spd* von *m/s* zu *knots* umgewandelt und mit *main.speed* als *topic* in den internen *State* geschrieben werden.

Auf jeder der genannten Ebenen kann eine Übertragung in beide Richtungen stattfinden. Wobei die jeweilige Ausgabe geeignet sein muss, um von der nächsten Ebene als Eingabe verwendet zu werden. Ist diese Voraussetzung erfüllt, kann die Gesamtübertragung bidirektional durchgeführt werden.

Die Koordination dieses dreiphasigen Prozesses übernimmt das zuvor strukturbezogen eingeführte *ProxyBehaviour*, im Sinne einer integrierten Variante eines *Gateway Programms*, wie [Schulze et al., 1999] es definiert haben. Die einzelnen Transformations- und Übertragungsschritte werden dabei durch Instanzen der Klassen *Dictionary*, *FormatAdapter* und *CommunicationAdapter* durchgeführt. Konfiguration und Ablauf des *Dictionary* werden im nächsten Abschnitt separat betrachtet. Um der Forderung nach Anbindbarkeit möglichst vieler gängiger Ausprägungen von Testobjekten gerecht zu werden (vgl. Anforderung *A1.2 Interoperabilität* und lösungsbezogene Anforderung *LA19*) geben das entworfene Metamodell und die auf diesem aufbauende generische Simulationsanwendung keine konkreten Abläufe oder innere Strukturen für die beiden Adapter vor. Stattdessen müssen konkrete *Format-* und *CommunicationAdapter*-Varianten als Teil des mehrschichtigen Modells (vgl. Abschnitt 14.1.3) für die Nutzung in einer Szenarioinstanz bereitgestellt werden.

Der Übertragungsprozess wird von der Instanz des *ProxyBehaviour,* wie in Abbildung 89 dargestellt, koordiniert: Nach Aufruf der Schnittstellen-Methode *step* der *ProxyBehaviour*-Instanz durch den Fe-

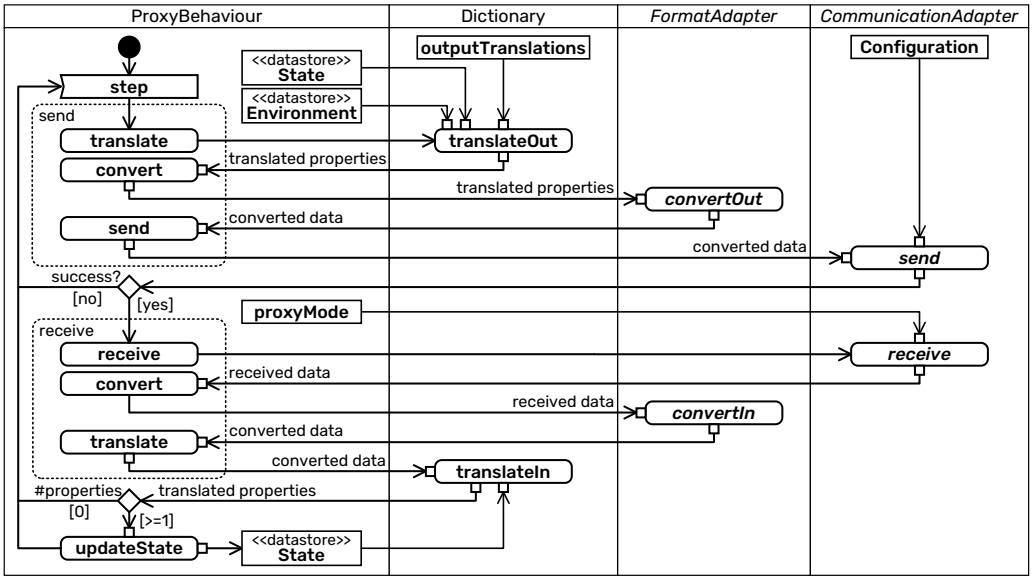


Abbildung 89: UML-Aktivitätsdiagramm der Kommunikation und Übersetzung durch ein Proxy-Verhalten

derate-Container werden nacheinander die auf ausgehende Kommunikation ausgerichteten Transformations- und Kommunikationsmethoden des *Dictionary*, des *FormatAdapter* und des *CommunicationAdapter* aufgerufen. Im Rahmen dessen werden die Zwischenergebnisse an den jeweils nächsten Methodenaufruf weitergereicht. Für eingehende Kommunikation ist die Reihenfolge der Aufrufe umgekehrt. Durch dieses zentral koordinierte, stringente Vorgehen müssen die Adapterimplementierungen jeweils nur eine Schnittstellenmethode pro Transformationsrichtung konkret umsetzen.

14.3.4.2 BIDIREKTIONALE INFORMATIONSTRANSFORMATION

Die Transformationen auf Ebene des Informationsmodells werden durch eine Instanz der konkreten Klasse *Dictionary* durchgeführt. Eine ableitende Implementierung einer abstrakten Klasse ist hier somit im Gegensatz zum *Format-* und *CommunicationAdapter* nicht nötig. Vielmehr erfolgt die Konfiguration durch die Zuweisung von *ExternalValue-* und *Translation*-Instanzen im Rahmen der Szenariospezifikation (vgl. Abschnitt 14.3.2.5). Eine solche Konfiguration enthält dabei Angaben zu den Merkmalen der zu verwendenden externen Werte und setzt diese durch die Angabe der jeweiligen Pfade, wie sie im vorigen Abschnitt beispielhaft genannt wurden, in Verbindung mit Werten der zentralen *State-* und *Environment*-Instanzen.

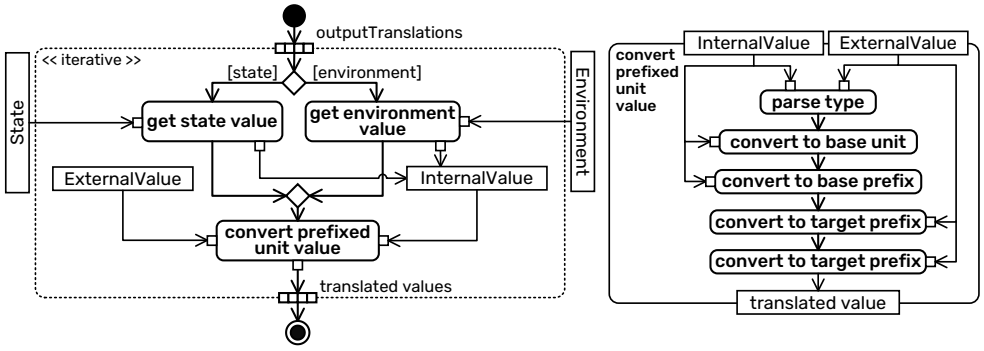


Abbildung 90: UML-Aktivitätsdiagramm des Ablaufs der Konvertierung von Werten, bevor diese ein externes Textobjekt kommuniziert werden, entsprechend der als Teil des Szenarios spezifizierten *OutputTranslation*s.

Abbildung 90 zeigt den Ablauf der durch das *Dictionary* durchgeführten Konvertierung eines internen Wertes, angestoßen durch den Aufruf der Methode *translateOut* einer *Dictionary*-Instanz während des Fortschreitens in der Simulationszeit (vgl. Abbildung 89). Der Konvertierungsprozess geschieht ebenfalls in mehreren aufeinanderfolgenden Schritten: (1) Entsprechend des angegebenen *path* wird die Referenz zum internen, zu konvertierenden Wert identifiziert; (2) wenn möglich, wird eine Kopie des referenzierten internen Wertes zu dem konfigurierten Datentyp des externen Wertes konvertiert; (3) handelt es sich um einen numerischen Wert, wird dieser entsprechend der Ziel-*Unit,* dem Ziel-*Prefix* und dem angegebenen Offset umgewandelt. Die Umwandlung ist durch die Klassen *Unit* und *Prefix* sowie die standardmäßige automatische Instanziierung aller SI-Basiseinheiten und SI-Präfixe durch einfache Umrechnung anhand der Angaben der Ausgangs- und Ziel-Einheiten und -Präfixe möglich:

$$unprefixedSourceValue = sourceValue \cdot \left(10^{soucePrefixPower}\right)$$

$$baseValue = \left((unprefixedSourceValue + sourcePreOffset) \cdot sourceFactor\right) + sourcePostOffset$$

$$targetUnitValue = \left(\frac{baseValue - targetPostOffset}{targetFactor}\right) - targetPreOffset$$

$$prefixedTargetValue = targetUnitValue \cdot \left(10^{-targetPrefixPower}\right)$$

$$\underline{translatedValue} = prefixedTargetValue + targetOffset$$

14.3.4.3 ORCHESTRIERUNG

Obwohl die verteilte Ausführung einer Simulationsanwendung in der Regel mit einem erheblichen Leistungsmehraufwand für die Kommunikation zwischen den einzelnen Simulatoren verbunden ist, ist es unbestreitbar, dass sie für viele hier betrachtete Anwendungsfälle notwendig ist. Dazu gehören unter anderem die Erfüllung von Lizenz- und Softwareanforderungen, der Zugriff auf vor Ort nicht unterstützte Plattformen und Laufzeitumgebungen sowie die Gewährleistung der sicheren Kommunikation mit einem externen Testobjekt innerhalb einer von der eigenen Infrastruktur abgetrennten Umgebung [Hatledal et al., 2019]. Wenn die Performanz der Gesamtsimulation eine Priorität ist und das zu simulierende Szenario eine Vielzahl an Simulationsteilnehmern enthält, sollte es zudem möglich sein, die einzelnen Teilsimulationen, wie bereits in Abschnitt 14.1.7 „Das Testobjekt als Teilnehmer am Simulationslauf“ angeschnitten wurde, auf mehrere Rechenknoten zu verteilen und parallel auszuführen, um so die Rechenlast effektiv zu verteilen und Skalierbarkeit zu gewährleisten.

Die lösungsorientierte Anforderung LA08 fordert eine einfache Ausführung des Simulationslaufs auf Basis einer Szenarioinstanz als Eingabe für die Simulationsanwendung. Diese Anforderung gilt selbstverständlich auch, oder vor allem, dann, wenn die Simulation vollständig oder teilweise physisch verteilt ausgeführt werden soll, um den entstehenden Mehraufwand zumindest aus Nutzersicht möglichst gering zu halten.

Bemühungen dahingehend, eine HLA-Simulation *on the grid* auszuführen, also die Kommunikation flexibel über das Internet zu ermöglichen, gibt es schon einige Jahre. Pan et al. haben bereits [2007] eine Architektur für eine serviceorientierte flexible Verteilung vorgestellt. Derartige Ansätze unterscheiden sich aber von dem hier erarbeiteten Konzept dadurch, dass in dem vorliegenden Fall alle Simulationsteilnehmer auf demselben Modell basieren und der Simulationslauf vollständig auf der Grundlage einer einzelnen Szenarioinstanz initialisiert wird. Somit haben alle Federates einen identischen Lebenszyklus und ein dynamisches Ein- oder Austreten von Simulationsteilnehmern ist weder vorgesehen noch nötig. Stattdessen besteht die Notwendigkeit, möglichst unabhängig von dem konkret ausführenden Computersystem zu sein und möglichst wenig Anforderungen an die konkrete Softwarekonfiguration des ausführenden Computersystems zu stellen. Dies ist wünschenswert, um den nötigen Aufwand und die Nutzungshürde gering zu halten (vgl. Begründung von Teilziel 2).

Der gewählte Ansatz setzt daher auf eine zentrale Orchestrierung auf Basis von Containerisierung. Das Verfahren der Containerisierung hat als leichtgewichtige Alternative zur Virtualisierung in den letzten 10–15 Jahren vorwiegend durch den breiten Einsatz von Microservice-basierten Architekturen an Popularität gewonnen. Einige Vorteile von Containerisierung gegenüber klassischer Virtualisierung sind die gemeinsame Nutzung des Kernels des Host-Betriebssystems, ein geringerer Overhead und kürzere Startzeiten bei weiterhin vorhandener Isolation der Container-Inhalte vom Host-System. [Bentaleb et al., 2022; Watada et al., 2019] Ein sogenannter *Application Container* [Verma et al., 2022] ist dabei ein individueller Container, der für den Zweck erstellt wurde, eine einzelne Anwendung flexibel auf unterschiedlichen Host-Systemen ausführen zu können. Für diesen Zweck werden die Softwareanwendungen mitsamt aller Abhängigkeiten, wie Bibliotheken, Ausführungsumgebungen und Konfigurationsdateien, zu einer isolierten Einheit, dem Container, verpackt. Die Menge der Voraussetzungen, die ein entferntes Host-System erfüllen muss, reduziert sich so auf die notwendige Möglichkeit, von außerhalb des lokalen Netzwerkes (abgesicherten) Zugriff zu gewähren, und je nach konkreter Implemen-

tierung des Containerisierungsprinzips ggf. eine entsprechende für den Betrieb der Container nötige Softwareanwendung vorab zu installieren.

Der Ablauf der Initialisierung eines verteilten Simulationslaufs auf Basis einer Szenarioinstanz unter Nutzung von Containerisierung ist in Abbildung 91 dargestellt: (1) Die zentrale Verwaltungskomponente der Simulationsanwendung liest die Szenarioinstanz ein; (2) Eine Modellbibliothek wird entsprechend der Angaben im Konfigurationsteil der Szenarioinstanz identifiziert, bei Bedarf übertragen, eingelesen; (3) Die Szenarioinstanz wird unter Verwendung der Klassen der Modellbibliothek interpretiert, d. h. die in der Szenarioinstanz spezifizierten Objekte werden instanziiert; (4) Sind im Konfigurationsteil der Szenarioinstanz Angaben zu einer Verteilten Ausführung vorhanden, wird ein Client-Container erzeugt, der die aktuell geladene Modellbibliothek, die Simulationsanwendung, eine HLA-Implementierung, und weitere Abhängigkeiten umfasst; (5) Der Container wird in *Repository,* d. h. eine zentrale Datenhaltung hochgeladen; (6) Der Verteilungskonfiguration entsprechend wird (6.1) die Federation erzeugt, eine Verbindung zu den konfigurierten entfernten Host-Systemen aufgebaut und (6.2.1—6.2.n) dort der Befehl zur Ausführung der Szenarioinstanz durch die containerisierte Simulationsanwendung gegeben; (7) Der im Startbefehl angegebene Container wird von den Host-Systemen aus dem zentralen Repository heruntergeladen und anschließend gestartet.

Da der zentrale Verwaltungsteil keine Kontrolle über die Dauer des Startprozesses der einzelnen entfernten Hosts hat, wird auf eine Rückmeldung aller Hosts bzgl. ihrer Bereitschaft zur Ausführung des Simulationslaufs gewartet. Anschließend sendet der zentrale Verwaltungsteil das Signal zum Starten des Simulationslaufs an die RTI und entsprechend an alle Simulationsteilnehmer. Hierfür wird die HLA-Funktionalität der *SynchronizationPoints* genutzt.

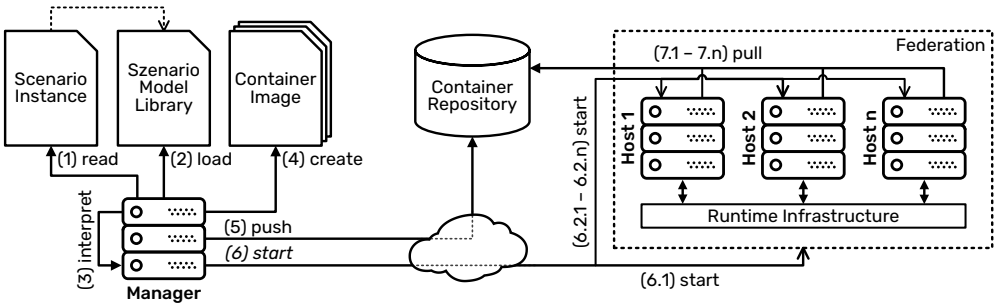


Abbildung 91: Ablauf der Container-basierten Orchestrierung der verteilten Simulation

15 EXPERIMENTELLE UMSETZUNG

Der Entwurf in Form inhaltlich und funktional ausgestalteter Lösungskonzepte wurde im nächsten Schritt prototypisch umgesetzt, um eine belastbare Grundlage für die anschließende Evaluation der konzipierten Lösung zu schaffen (vgl. Abschnitt *Forschungsdesign*). Eine experimentelle Umsetzung ist Bestandteil eines typischen ingenieurwissenschaftlich-konstruktiven Vorgehens, in dem ein Entwurf durch ein prototypisch umgesetztes technisches Artefakt konkretisiert und überprüfbar gemacht wird. Der Prototyp fungiert somit als Brücke zwischen konzeptioneller Ausarbeitung und empirischer Untersuchung: Er operationalisiert die lösungsorientierten Entwurfsentscheidungen in einer lauffähigen Realisierung und macht sie so für experimentelle Untersuchungen zur Überprüfung der Zielerreichung zugänglich. Damit die Ergebnisse der nachfolgenden Evaluation nachvollziehbar eingeordnet werden können, werden im Folgenden einige wenige wesentliche Umsetzungsentscheidungen betrach-

tet, der verwendete Technologie-Stack[79] begründet beschrieben und die entstandenen softwaretechnischen Artefakte einordnend vorgestellt.

Das Ergebnis der experimentellen Umsetzung ist das Projekt *Distributed Component-Based Traffic Simulation* (DisCo-BaTS)[80]. Die entstandenen softwaretechnischen Artefakte sowie der vollständige Quelltext wurden gemeinsam unter diesem Namen aufgeteilt auf mehrere Repositorien (vgl. Abbildung 92) auf der öffentlichen Softwareentwicklungsplattform *GitHub*[81] veröffentlicht. Über das reine Studieren der Inhalte hinaus sind die Nutzung, Anpassung, Vervielfältigung und Verbreitung entsprechend der Lizenzbedingungen der *GNU Lesser General Public License version 3* (LGPLv3)[82] gestattet.

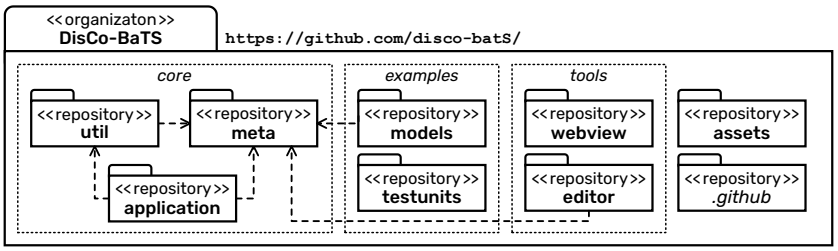


Abbildung 92: Repository-Struktur der öffentlich zugänglichen Ergebnisse

Die Repositorien *meta*, *simulation* und *util* umfassen die Umsetzungen der vorgestellten Kernkonzepte und bilden somit gemeinsam den zusammenfassend als *core* bezeichneten funktionalen Kern von *DisCo-BaTS*. Dabei ist die vollständige Implementierung des in Abschnitt 14.3.2 vorgestellten Metamodells im Repositorium *meta* verortet. Das Repositorium *simulation* umfasst die prototypische Implementierung der funktionalen Konzepte, die in den Abschnitten 14.3.3 und 14.3.4 vorgestellt wurden.

Im Rahmen der Evaluationsdurchführung sowie weiterer experimenteller Einsätze der *DisCo-BaTS* außerhalb des Betrachtungsrahmens dieser Arbeit wurden gemäß der in den Abschnitten 14.1.2–14.1.4 und 14.1.5.1 vorgestellten Konzepte beispielhafte hierarchische Modelle ausgestaltet. Dem in Abschnitt 14.1.5.3 vorgestellten Lösungskonzept entsprechend wurden basierend auf diesen beispielhafte Modellbibliotheken definiert. Diese bilden den Inhalt des Repositoriums *models.* In die Gruppe der Beispiele (engl.: *examples*) ordnen sich zusätzlich die beispielhaften Implementierungen zweier einfacher externer Testobjekte ein. Diese wurden im Rahmen der Evaluation entwickelt und eingesetzt (vgl. Abschnitt *Erprobung der Interoperabilität*), sind aber auch darüber hinaus zu Testzwecken nutzbar. Das Repositorium *assets* umfasst kleinere Mengen beispielhafter Konfigurationsdateien, Templates, Skripte etc. Während der Erprobung der iterativen Entwicklungsstände und einiger experimenteller Einsätze der (Teil-)Ergebnisse sind zudem zwei, hier unter dem Namen *tools* zusammengefasste Anwendungen entstanden, die in Verbindung mit der Kernanwendung (*core*) genutzt werden können. Diese demonstrieren somit die Möglichkeit, zusätzliche Tools und Werkzeuge in das entstandene Ökosystem zu integrieren. Konkrete enthält das Repositorium *editor* ein rudimentäres Softwarewerkzeug zur vereinfachten Erstellung und Bearbeitung von Szenarioinstanzen unter Nutzung der entsprechenden Modellbibliothek unter Verwendung einer einfachen grafischen Benutzeroberfläche. Das Repositori-

[79] Der Begriff *Technologie-Stack* bezeichnet in der Regel die Gesamtheit der Programmiersprachen, Frameworks, Bibliotheken und Tools, die bei der Entwicklung einer Anwendung zum Einsatz kommen.

[80] GitHub – dvdrhr, *DisCo-BaTS*: `https://github.com/DisCo-BaTS/` [letzter Zugriff: 17.02.26]

[81] GitHub, Inc., GitHub, *Build software better, together*: `https://github.com/` [letzter Zugriff: 17.02.26]

[82] Open Source Initiative, *GNU Lesser General Public License version 3*: `https://opensource.org/license/lgpl-3-0` [letzter Zugriff: 17.02.26]

um *webview* enthält den Programmcode einer Server- und einer Webanwendung. Der Server-Teil bietet einen Websocket an, welcher eingehende Nachrichten entgegennehmen und anschließend die TCP-Verbindung zum Absender (Client) halten kann. Anschließend können Daten von den verbundenen Clients empfangen werden. Empfangene Daten werden an alle verbundenen Clients weitergeleitet. Dieser einfache Mechanismus wird genutzt, um von einem entsprechend konfigurierten *GlobalObserver* aktuelle Zustandsdaten eines Simulationslaufs entgegenzunehmen und an die Webanwendung weiterzuleiten. Die Webanwendung zeigt entsprechend mittels einer einfachen grafischen Oberfläche die aktuelle Simulationszeit, Schiffe und die von diesen bisher zurückgelegte Strecke, Wasserkörper sowie Bojen an ihrer aktuellen Position auf einer Seekarte an, was eine grundlegende Beobachtung des Simulationslaufs zur Laufzeit ermöglicht. Ausschnitte der grafischen Oberfläche der Webanwendung sind in Abschnitt 16.2 zur Veranschaulichung der Testfälle genutzt worden.

Aufgrund dessen, dass die Gruppe der *core*-Artefakte die Umsetzung der Lösungskonzepte darstellt und somit den zu evaluierenden Anteil der Gesamtmenge entstandener Artefakte, werden diese nachfolgend genauer bezüglich der konkreten Auswahl genutzter Technologien und sich daraus ergebender Besonderheiten betrachtet. Diese technische Betrachtung ist beabsichtigt kurzgehalten, da die Umsetzung inhaltlich dem zuvor ausführlich vorgestellten Lösungsentwurf entspricht. In Kombination mit den öffentlich verfügbaren vollständigen Quelltexten aller Umsetzungsergebnisse ist die Nachvollziehbarkeit in ausreichendem Maße gegeben.

15.1 SIMULATIONSANWENDUNG

Die Entscheidung dafür, die Simulationsanwendung entsprechend der High Level Architecture (HLA) zu gestalten, hatte weitreichenden Einfluss auf die konkrete Ausgestaltung der übergreifenden Lösungskonzepte (vgl. Abschnitt 14.3). Da der entsprechende Standard [IEEE, 1516:2010] keine Referenzimplementierung umfasst und auch im Rahmen der Beschreibung der Schnittstellen, generellen Abläufe und zu verwendenden Datenstrukturen keinerlei Vorgaben bezüglich einer Implementierung macht, mussten zunächst entsprechende Implementierungen identifiziert, vergleichend untersucht und die begründete Entscheidung für die Nutzung einer dieser getroffen werden. Da diese Wahl wiederum ebenfalls weitreichende Auswirkungen auf den Großteil der technologischen Aspekte einer Simulationsanwendungsumsetzung hat, wurde diese mit der höchsten Priorität getroffen. Weitere technologische Entscheidungen bezüglich der experimentellen Umsetzung folgten entsprechend dieser primären Entscheidung. Dementsprechend wird nachfolgend ebenfalls zunächst die Umsetzung des grundlegenden Simulationsablaufs betrachtet und erst im Anschluss das eigentlich zentraler einzuordnende Metamodell.

Die Menge der verfügbaren RTI-Implementierungen ist überschaubar. Näher betrachtet und für den Einsatz als Grundlage für die experimentelle Umsetzung in Betracht gezogen wurden diejenigen, die in den Überblick-Arbeiten [Gütlein et al., 2020; Huiskamp & van den Berg, 2016; Malinga & Le Roux, 2009] gelistet sind. Eine eigene Recherche förderte keine weiteren, in diesen Arbeiten nicht genannten, öffentlich zugänglichen Implementierungen zutage.

Die näher untersuchten Implementierungen verfolgen jeweils einen von zwei existierenden architekturellen Ansätzen zur Umsetzung der Vorgaben des HLA-Standards (vgl. [Dahmann et al., 1998]): eine zentrale und eine dezentrale Implementierung der RTI. Die erste Variante ist die naheliegendere, da

die übliche abstrakte Darstellung einer Federation zumeist, auch im Rahmen der bisherigen Abschnitte dieser Arbeit, diese Anordnung zeigt: Die RTI existiert als ein einzigartiges, eigenständiges, Federation-zentrales Element, mit dem alle Federates der Federation kommunizieren (vgl. Abbildung 93, rechts). In der zweiten Variante ist die RTI keine eigenständige zentrale Instanz, sondern wird durch eine Menge von lokalen RTI-Instanzen, die jeweils einem Federate zugeordnet sind, gebildet. Diese verteilten RTI-Instanzen kommunizieren miteinander, um die Aufgaben einer RTI erfüllen zu können (vgl. Abbildung 93, links). Der Datenaustausch zwischen den lokalen RTIs kann z. B. in lokalen Netzwerken mittels Multicast-Kommunikation realisiert werden.

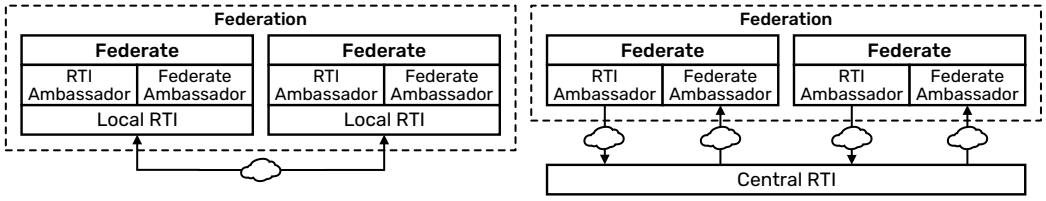


Abbildung 93: Dezentrale (links) und zentrale (rechts) Implementierung einer RTI

Für den Vergleich und die anschließende Auswahl einer der RTI-Implementierungen für den Einsatz im Rahmen der experimentellen Umsetzung wurden folgende Anforderungen festgelegt:

Open Source: Der Quelltext der Implementierung sollte öffentlich einsehbar sein und sowohl die Nutzung als auch die Anpassung sollten möglich sein.
Dezentrale RTI: Eine dezentrale RTI-Implementierung wird bevorzugt, da dies die Eigenständigkeit der Federates zusätzlich betont und stärkt. Durch die geringere Menge der Kommunikationswege und die entfallene Notwendigkeit, eine zentrale RTI-Anwendung so im Netzwerk zu platzieren, dass sie von allen Federates erreicht werden kann, verspricht dieser Ansatz, der flexiblere zu sein in Bezug auf eine verteilte Ausführung, die über die Grenzen eines lokalen Netzwerkes hinausgeht.
Aktualität & Support: Das Produkt bzw. das Projekt muss aktiv gepflegt und aktualisiert werden sowie eine aktive Nutzercommunity oder eine andere Möglichkeit für Hilfestellungen bieten.
HLA-Standard: Da ein zentraler Teil des ausgestalteten Konzeptes auf die Verwendung von modularen FOMs setzt, muss die RTI-Implementierung mindestens den IEEE-Standard 1516 in der Version 2010 unterstützen.

Die untersuchten RTI-Implementierungen und ihre Ausprägungen der genannten Merkmale sind in Tabelle 15 zusammenfassend genannt. Portico (Esw.: poRTIco) erfüllt alle gesetzten Anforderungen: Die aktuelle stabile Version 2.1.3 wurde am 4. Mai 2025 veröffentlicht, die letzte Änderung am noch nicht als stabil markierten Zwei der Version 2.2.x wurde am 20. Januar 2026 hinzugefügt – das Projekt wird sehr aktiv gepflegt. Auch die Community ist aktiv und die Projektverantwortlichen antworten regelmäßig auf Anfragen und Fehlermeldungen: Der letzte Beitrag eines Nutzers bei GitHub ist auf den 20. Januar 2026 datiert. Portico umfasst eine Implementierung annähernd aller vom HLA-Standard definierten Schnittstellenmethoden – einige werden in der Version 2.1.3 noch nicht unterstützt, diese sind aber vorerst nicht von Relevanz für die Umsetzung der Konzepte. Portico setzt auf einen dezentralen Ansatz und stellt zu diesem Zweck die *Local RTI Component (LRC)* zur Verfügung.

Für die Integration der IEEE 1516:2010-Implementierung Porticos stehen Schnittstellen für die Programmiersprachen Java und C++ bereit. Für die Implementierung von Portico selbst wurde Java ge-

Tabelle 15: Merkmale verschiedener RTI-Implementierungen

Name	Version	HLA-Standard	Anbindungen	Art	Lizenz
Pitch pRTI[83]	6	IEEE 1516-2025	Java, C++, Netzwerk	zentral	Closed Source, kommerziell
MAK RTI[84]	5.0.1	IEEE 1516-2010	C++, (Java)	zentral / verteilt	Closed Source, kommerziell
CERTI[85]	3.5.1	IEEE 1516-2010 (teilw.)	C++, (Java)	zentral	GPL & LGPL
poRTIco[86]	2.1.3	IEEE 1516-2010 (teilw.)	Java, (C++)	verteilt	CDDL, Version 1.0
OpenHLA[87]	0.6.1	IEEE 1516-2010 (teilw.)	Java	zentral	Apache License, Version 2.0

nutzt. Die C++-Schnittstelle wird durch einen Wrapper um die Java-Schnittstelle zur Verfügung gestellt: Im Hintergrund wird eine *Java Virtual Machine* (JVM) gestartet, welche die Java-Schnittstelle initialisiert und die vom Wrapper durchgereichten Aufrufe ausführt. Zur Vermeidung von Ressourcen-Mehraufwand, zusätzlicher Komplexität und zusätzlicher potenzieller Fehlerquellen wurde für die experimentelle Umsetzung der *core*-Komponenten ebenfalls Java[83] als Programmiersprache gewählt.

Der resultierende Aufbau einer beispielhaften *DisCo-BaTS*-Federation ist in Abbildung 94 dargestellt. Zu erkennen sind die unterschiedlichen Federate-Typen (vgl. Abschnitt 14.3.3), die jeweils zur Erfüllung verschiedener Aufgaben einen unterschiedlichen Funktionsumfang besitzen. Zusätzlich zu den konzeptionell vorgestellten Typen ist ein sogenanntes *MasterFederate* Teil eines Simulationslaufs. Dieses wird als erstes Federate gestartet und hat die Aufgabe die Federation zu erstellen sowie den Beginn des eigentlichen Simulationslaufs zu synchronisieren: (1) Nachdem das *MasterFederate* erzeugt, initialisiert und gestartet wurde, erzeugt es die Federation; (2) Anschließend werden die restlichen *Federates* initialisiert und gestartet, dabei tritt jedes als letzter Schritt des Startvorgangs der Federation bei; (3) Das *MasterFederate* wartet bis alle weiteren Federates der Federation beigetreten sind und dies über eine Mitteilung des Erreichens des *SynchronizationPoints READY_TO_RUN* an die RTI und somit die gesamte Federation bekannt gegeben haben; (4) Als letztes bestätigt auch das *MasterFederate* bekannt, dass es den *SynchronizationPoint* erreicht hat, wodurch allen Federates der Federation mitgeteilt wird, dass die Federation synchronisiert ist, woraufhin Federates in den Ausführungsstatus *running* wechseln und mit der Ausführung ihrer implementierten Simulations-Schleife beginnen.

Der Lebenszyklus eines *SimulationFederate* ist in Abbildung 95 dargestellt. Die Lebenszyklen der anderen umgesetzten Federate-Typen stellen überwiegend Teilmengen des dargestellten Lebenszyklus dar, da diese einen geringeren Funktionsumfang besitzen. Daher wird hier auf eine explizite Darstellung

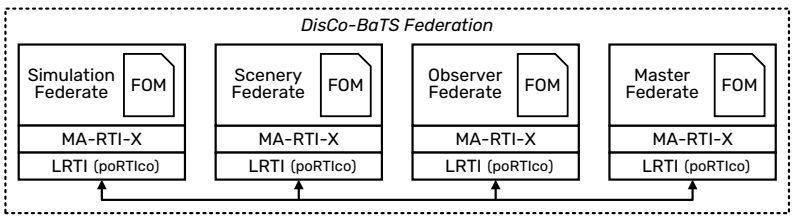


Abbildung 94: Jeweils eine eigenständige Instanz der lokalen RTI-Implementierung Porticos (LRTI) und der entwickelten *Model-Aware-RTI-Exchange* (MA-RTI-X) Adapterschicht ist Teil jedes Federates.

[83] Zum Zeitpunkt des Beginns der finalen experimentellen Umsetzung war das Java Developer Kit (JDK) in der Version 21 die aktuelle Long-Term-Support-(LTS-)Version. Die Umsetzung wurde daher mit Oracle OpenJDK in der Version 21.0.1 begonnen, welche später durch die Version 21.0.2 ersetzt wurde: `https://openjdk.org/projects/jdk/21/` [letzter Zugriff: 17.02.26]

[84] BAE Systems OneArc, *Introducing Pitch pRTI 6*: `https://onearc.com/news/introducing-pitch-prti6/` [letzter Zugriff: 17.02.26]

[85] MAK Technologies, Inc., *MAK RTI*: `https://www.mak.com/mak-one/tools/mak-rti` [letzter Zugriff: 17.02.26]

[86] Savannah – ISAE-SUPAERO, *CERTI*: `https://savannah.nongnu.org/projects/certi` [letzter Zugriff: 17.02.26]

[87] GitHub – OpenLVC, *Portico*: `https://github.com/openlvc/portico` [letzter Zugriff: 17.02.26]

[88] SourceForge – Michael Newcomb, *Open HLA*: `https://sourceforge.net/projects/ohla/` [letzter Zugriff: 17.02.26]

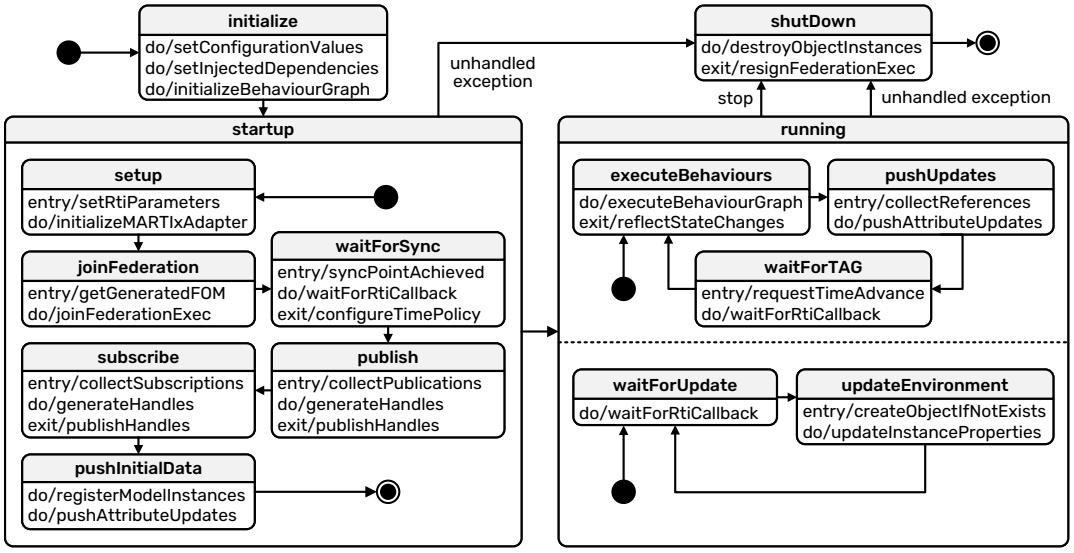


Abbildung 95: Lebenszyklus eines *SimulationFederate* als Container eines *Reactors*

dieser verzichtet. Viele der angedeuteten funktionalen Abläufe der einzelnen Phasen werden von der *Model-Aware-RTI-Exchange*-Schicht übernommen. Die Abgrenzung der Vermittlungsschicht vom Federate-Container ist an dieser Stelle zugunsten des Überblicks nicht explizit dargestellt. Die Implementierung der Inhalte der Schicht entspricht dem in Abbildung 83 dargestellten Entwurf. Die Verbindung zu Portico als RTI-Implementierung wird dabei durch Nutzung der durch diese zur Verfügung gestellten Interfaces und Klassen geschaffen: Als RTI-Ambassador wird während der Initialisierungsphase der *SimulationFederate*-Instanz mit Hilfe einer Instanz der Portico-Klasse *Rti1516eFactory* eine Referenz auf eine Instanz der bereitgestellten Klasse *Rti1516eAmbassador* erzeugt. Diese implementiert die vom IEEE-Standard 1516:2010 vorgeschriebenen Schnittstellenmethoden eines RTI-Ambassadors. Der *FederateAmbassador* hingegen stellt eine Eigenimplementierung dar, die zwar vom *NullFederateAmbassador* Porticos erbt und somit die Schnittstelle *FederateAmbassador* implementiert, aber die relevanten Methoden überschreibt.

15.2 MODELLE

Das Metamodell wurde entsprechend des in Abschnitt 14.3.2 vorgestellten Konzeptes in Java-*Interfaces* und konkrete sowie abstrakte -*Classes* überführt. Die Struktur des Unterpakets *scenario* orientiert sich an den Beziehungen der möglichen Inhalte eines Szenarios, wie sie in Abbildung 47 dargestellt wurde. Das Paket *dto*[89] enthält Klassen, die lediglich Informationen umfassen, die für die Initialisierung des Simulationslaufs genutzt werden. In diese Kategorie fallen u. a. *Scenario*, *Library* und *ObservedType*.

Mit Hilfe des Build-Management- und Projektverwaltungswerkzeugs *Apache Maven*[90], das für alle umgesetzten Teilprojekte eingesetzt wird, und des *Apache Maven JAR Plugin*[91] kann aus den umgesetzten Klassen ein *Java Archive* (JAR) erzeugt werden, welches das gesamte Metamodell gebrauchsfertig in Form einer einzelnen Datei umfasst und als Grundlage für die Umsetzung konkreter Ausprägungen der konzipierten mehrschichtigen (Teil-)Modelle genutzt werden kann.

[89] *DTO* ist eine gebräuchliche Abkürzung für das Konzept eines *Data Transfer Objects*.
[90] The Apache Software Foundation, *Welcome to Apache Maven*: `https://maven.apache.org/` [letzter Zugriff: 17.02.26]
[91] The Apache Software Foundation, *Apache Maven Compiler Plugin*: `https://maven.apache.org/plugins/maven-jar-plugin/` [letzter Zugriff: 17.02.26]

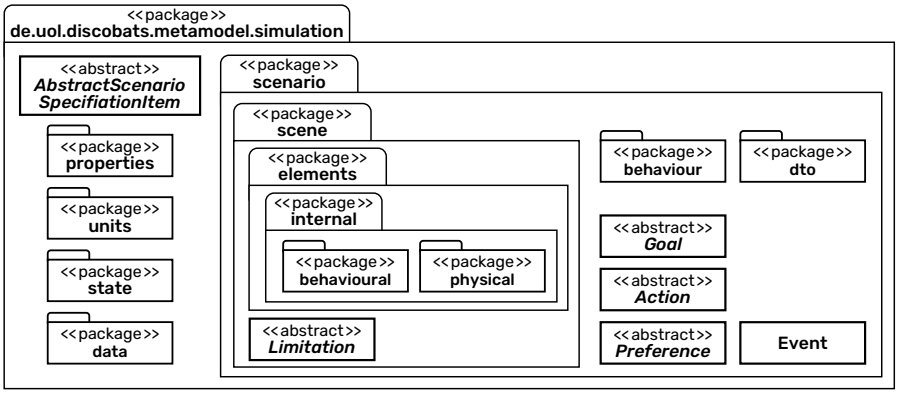


Abbildung 96: Paketstruktur der Implementierung des Metamodells

15.2.1 HIERARCHISCHE SZENARIOMODELLE

Aufbauend auf der exportierten Metamodell Implementierung können Szenariomodelle entsprechend der Konzepte aus den Abschnitten 14.1.3, 14.1.5.1 und 14.1.5.3 durch Vererbung erstellt werden. Die beispielhafte Umsetzung einer konkreten eingebetteten Modellhierarchie schlägt zu diesem Zweck die Verwendung von geschachtelten Maven-Projekten vor. Maven ermöglicht die Schachtelung von Projekten durch die Deklaration der inneren Projekte als sogenannte *modules* der umfassenden Projekte. Die entsprechende Projektstruktur und die Vererbungsbeziehungen der enthaltenen Modellklassen sind in Abbildung 96 dargestellt.

Jede der Ebenen O_2, O_1 und O_0 sowie jedes der Teilmodellpakete D_1, D_2 und D_3 liegt in Form eines eigenen Maven-Projekts vor und kann entsprechend, wie auch bereits das Metamodell, zusammengefasst als Java-Archiv (JAR) exportiert werden. Diese wird wiederum von dem Projekt, das die nächsthöhere Schicht repräsentiert, als Abhängigkeit importiert. Entsprechend ist die Gestaltung derartiger Modellhierarchien flexibel und die einzelnen Schichten sind in hohem Maße wiederverwendbar.

15.2.2 PERSISTENTE SZENARIOINSTANZEN

Das als JAR-Datei vorliegende Ergebnis des Exports eines Moduls der letzten Schicht O_0 (vgl. *o0-testcase* und *exampleLib* in Abbildung 96) stellt im Sinne des erarbeiteten Konzeptes die Modellbibliothek dar (vgl. 14.1.5.3). Die Top-Level-Modellelemente dieser Szenariobibliothek können nun entsprechend des Lösungskonzepts parametriert werden (Anforderung LA15) und zu einer konkreten Szenarioinstanz zusammengesetzt werden. Eine solche Szenarioinstanz, genauer gesagt ihre beschreibende Repräsentation, muss entsprechend der lösungsbezogenen Anforderung LA16 dabei in einer persistierbaren serialisierten Form vorliegen (vgl. Abschnitt 14.3.3). Im Rahmen der experimentellen prototypischen Umsetzung wurde dafür die Extensible Markup Language (XML) gewählt. XML ist ein Standard für die Übertragung von strukturierten Daten im Internet, kann dementsprechend äußerst flexibel auf die konkreten Bedürfnisse angepasst und zur Beschreibung jeder Art von Daten genutzt werden [Banzal, 2020]. XML-Dokumente sind zudem sowohl maschinenlesbar als auch in einem häufig ausreichenden Maß menschenlesbar. Aufgrund der Möglichkeit, mit XML geschachtelte Datenstrukturen zu repräsentieren, ist das Format zudem gut geeignet, um Elemente ontologischer Datenstrukturen zu repräsentieren [Haw et al., 2023]. Das Wurzelelement einer XML-Beschreibung einer Szenarioinstanz bildet die XML-Repräsentation des Elements *Scenario* des vorgestellten Metamodells (vgl. Abbildung 80). Ein unvollständiger, beispielhafter Rumpf eines entsprechenden persistierbaren Szenarioinstanz-XML-Dokuments ist in Anhang A gelistet.

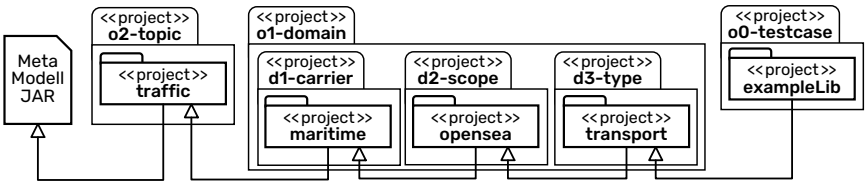


Abbildung 97: Realisierung des hierarchischen Modells durch Abhängigkeiten zwischen Maven-Modulen

15.2.3 TRANSITIVES SIMULATIONSMODELL

Zur Überführung eines XML-Dokuments, wie es im vorangegangenen Abschnitt beschrieben wurde, im Sinne eines Unmarshalling-Prozesses, wie er in Abschnitt 14.3.3.1 *Modellüberführung* vorgestellt wurde, wurde die *Jakarta XML Binding* (JAXB)[92] Programmierschnittstelle genutzt. Konkreter wurde die Implementierung *com.sun.xml.bind:jaxb-impl* in der Version 4.0.6[93] in das *meta*-Projekt integriert. Aufgrund der Abhängigkeitsbeziehungen der Teilprojekte erben weitere Projekte diese Integration.

Die JAXB-Implementierung ist in der Lage, XML-Dokumente zu lesen und die enthaltenen XML-Elemente in entsprechende Java-Objektinstanzen zu überführen. Voraussetzung dafür ist, dass der JAXB-Implementierung die Java-Klassen und ihre Einordnung innerhalb der Paketstruktur (*classpath,* dt.: Klassenpfad) bekannt sind. Aufgrund der hochflexiblen Modellierungshierarchie, zugunsten der Wiederverwendbarkeit von Teilmodellen, können diese Pfade nicht im Voraus bestimmt werden und sich zudem zwischen einzelnen Teilmodellpaketen voneinander unterscheiden. Daher wurde ein Maven-Plugin (im Maven Kontext als *MOJO* bezeichnet) umgesetzt, welches in den Build-Prozess von Bibliotheksprojekten (d. h. untergeordnete Projekte des Projekts *o0-testcase*, vgl. Abbildung 97) vor der Ausführung des *JAR-Plugins* (vgl. Einleitung dieses Abschnitts) eingehängt werden muss. Dieses Plugin durchläuft alle Java-Klassen des aktuellen Projekts, aller direkten Abhängigkeiten des aktuellen Projekts sowie aller transitiven Abhängigkeiten. Dabei werden die Klassenpfade auf das Vorhandensein des Präfixes *de.uol.discobats* sowie auf ein Vorkommen eines Pfadsegments *model* oder *metamodel* kontrolliert. Ist beides gegeben, dann wird der vollständige Klassenpfad dem Klassen-Index dieser Bibliothek hinzugefügt (vgl. Listing 3). Dieser Klassen-Index wird in Form einer Textdatei mit in die JAR-Datei der Bibliothek aufgenommen und von dort durch die Simulationsanwendung eingelesen und an die JAXB-Implementierung übergeben.

Eine weitere Voraussetzung dafür, dass es der JAXB-Implementierung möglich ist, die Inhalte des XML-Dokuments den korrekten Klassen und Datentypen zuzuordnen, ist z. T. die Markierung der Java-Klassen und -Felder durch das Hinzufügen von Annotationen. Um die Lesbarkeit der Modellklassen zu erhöhen und die Erstellung zu vereinfachen, wurden eigene Annotationen eingeführt, die u. a.

Listing 3: Ausschnitt eines während des Packaging der Bibliothek erzeugten Klassen-Index

```
metamodel.simulation.elements.internal.physical.Element
metamodel.simulation.elements.internal.behavioural.Reactor
metamodel.simulation.elements.DynamicReactor
metamodel.simulation.properties.Property
model.o2.traffic.elements.actors.TrafficParticipant
model.o1.d1.maritime.actors.Vessel
model.o1.d2.opensea.actors.SeagoingVessel
model.o1.d3.transport.actors.ContainerShip
model.o0.libraries.example.actors.ExampleContainerShip
```

[92] Eclipse Foundation, *Jakarta XML Binding 4.0*: https://jakarta.ee/specifications/xml-binding/4.0/ [letzter Zugriff: 17.02.26]

[93] Maven Repository, com.sun.xml.bind » jaxb-impl » 4.0.6: https://mvnrepository.com/artifact/com.sun.xml.bind/jaxb-impl/4.0.6 [letzter Zugriff: 17.02.26]

Listing 4: Ausschnitt eines während des Packaging der Bibliothek erzeugten Klassen-Index

```
@ScenarioSpecificationItem
public abstract class TrafficParticipant extends DynamicReactor {

    @ExchangeableProperty
    @Unit(Units.SPEED.METERS_PER_SECOND.class)
    protected MutableProperty<Double> speed;
```

Listing 5: CMD-Skript zum einfachen Start der Simulationsanwendung

```
.\START.CMD  \path\to\scenario.xml           // start into CLI interface & wait for user input
.\START.CMD  \path\to\scenario.xml  --direct // start the simulation run immediately
.\START.CMD  \path\to\scenario.xml  --debug  // start and provide remote debbuging socket on port 8000
```

mehrere Annotationen kapseln und entsprechend des Kontextes des Projekts benannt wurden. Einen Ausschnitt einer solchen Klasse mit entsprechenden Markierungen durch projekteigene Annotationen ist in Listing 4 zu sehen.

Die JAXB-Implementierung kann lediglich auf Klassen zugreifen, die vom *Classpath* der aktuellen *Java Virtual Machine* (JVM) referenziert werden oder die dem *Classloader* (dt.: Klassenlader) der JVM bekannt sind. Da die Simulationsanwendung selbst, abgesehen von der Referenz auf das Metamodell, keine Modelle enthält, sind der ausführenden JVM die Modellklassen nicht bekannt. Zusätzlich kann die Referenz auf die zu nutzende Modellbibliothek erst während der Laufzeit geladen werden, da die entsprechenden Angaben Teil der einzulesenden XML-Datei sind, weshalb auch ein statisches Hinzufügen der JAR-Datei mittels eines Startparameters nicht sinnvoll möglich ist. Um entsprechend ein dynamisches Laden der innerhalb der XML-Datei referenzierten JAR-Datei enthaltenen Klassen zu realisieren, wird die *Instrumentation-API* eingesetzt. Diese wird von einer JVM bereitgestellt und ermöglicht es, Änderungen an dieser und an dem von ihr geladenen Bytecode zur Laufzeit vorzunehmen. Da der Zugriff auf diese API aus dem ausgeführten Programm selbst nicht möglich ist, wurde auf das Konzept der *Java-Agents* zurückgegriffen. Ein *Java-Agent* ist eine speziell erstellte JAR-Datei, die seit der Version 8 von Java an jede Ausführung einer anderen JAR angehängt werden kann und Zugriff auf die Instrumentation-API hat. Im konkreten Fall stellt die Simulationsanwendung selbst auch den Agent dar (die entsprechende JAR wird somit ein zweites Mal geladen, negative Auswirkungen sind aber nicht zu erwarten, da der Agent nur einmalig vor dem eigentlichen Programmstart aktiv ist). Der Agent wird beim Start der Anwendung statisch hinzugefügt, und die im JAR-Manifest hinterlegte *premain*-Klasse wird ausgeführt, bevor der Ablauf des eigentlich auszuführenden JARs beginnt. Die *premain*-Klasse ist hier die Klasse *LibraryLoader*, welche aus der als Startparameter übergebenen XML-Datei die Angaben zur Ortung der Modellbibliothek-JAR-Datei extrahiert und diese dem *Classloader* durch Aufruf der Instrumentation-API-Methode *appendToSystemClassLoaderSearch* bekannt macht. Anschließend wird durch die JVM der eigentliche Programmablauf initiiert.

Um die hinzugekommene Komplexität durch nötige Startparameter zu kapseln, wurde ein CMD-Skript erstellt, welches eine einfache Nutzung der Kommandozeilenschnittstelle zur Ausführung der Simulationsanwendung bietet. Einige Nutzungsbeispiele sind in Listing 5 abgebildet.

Der Programmablauf der Simulationsanwendung wird anschließend vom *SimulationManager* gesteuert (vgl. Abbildung 98). Dieser übergibt die Kontrolle zunächst an den *ScenarioConverter*, welcher mit Hilfe von JAXB die Inhalte des eingelesenen XML-Dokuments in die entsprechenden Java Objektinstanzen überführt. Mit Hilfe des *Interpreter*s werden anschließend die Inhalte der Szenarioinstanz be-

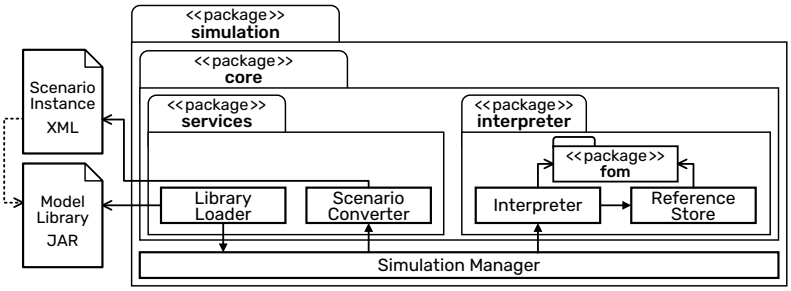


Abbildung 98: Kontrollfluss und Beziehungen der an der Interpretation einer Szenarioinstanz beteiligten Objekte

züglich ihrer Rolle / Bedeutung für den Simulationslauf vorbereitend verarbeitet (interpretiert). Im Rahmen dieses Vorgangs werden u. a. die Federate-Container initialisiert (vgl. „Modellüberführung“), die *FOM*-Module erzeugt (vgl. „Modelltransformation“) und der für die Kommunikation innerhalb der HLA-Federation zentrale *ReferenceStore* jeder Federate-Instanz entsprechend den diesem zugewiesenen *Element*-Instanzen befüllt (vgl. „Implizite Infrastrukturanbindung“*)*.

15.3 ORCHESTRIERUNG

Für die Umsetzung des auf Containervirtualisierung basierenden Konzeptes zur Realisierung einer zentralen, über das Szenarioinstanz-XML-Dokument konfigurierten Orchestrierung wurde *Docker*[94] genutzt. Docker ist die führende Software im Bereich der Containervirtualisierung [Rosa et al., 2023] und kann unter den drei populärsten Betriebssystemen Linux, macOS und Windows genutzt werden [Sobieraj & Kotyński, 2024]. Um die zugrunde liegende Komponente (Docker-Engine), welche Open Source ist, existiert ein umfangreiches Ökosystem, wie u. a. den in gewissem Umfang frei nutzbaren Docker-Hub als öffentliche Registry für Docker-Container[95]. Für die Ausführung von Federates auf Hosts in entfernten Umgebungen (*Remote Hosts*) wurde ein entsprechendes *Dockerfile* zur Erstellung des Container-Images erstellt. Der Inhalt des Dockerfiles ist in Anhang G aufgelistet. Images und Container auf Basis von Dockerfiles haben darüber hinaus den Vorteil, dass sie *Layering* und *Caching* nutzen/ermöglichen. Das heißt, dass jeder Befehl des Dockerfiles ein *Layer* darstellt und anhand einer Prüfziffer auf Veränderung geprüft werden kann. Wenn für einen Layer während des Erstellens des Images oder des Herunterladens des Images aus dem Repository keine Veränderung festgestellt wird, dann wird von diesem Layer eine zwischengespeicherte Variante genutzt. Wiederholtes Erstellen und Herunterladen von Images ist somit effizienter als der jeweils erste Durchlauf.

Listing 6: Konfiguration der Verteilung in der XML-Spezifikation einer Szenarioinstanz (gekürzt)

```
<distribution>
  <targets>
    <ssh alias="host1">
      <address>IP-ADDRESS/FQL-HOSTNAME</address>
      <username>USERNAME</username>
      <password>PASSWORD</password>
    </ssh>
  </targets>
  <assignments>
    <single>
      <id>exampleActor1</id>
      <targetMachine>host1</targetMachine>
    </single>
  </assignments>
</distribution>
```

[94] Docker Inc., *Docker: Accelerated Container Application Development*: https://www.docker.com/ [letzter Zugriff: 17.02.26]

[95] Das im Rahmen der Evaluation erstellte öffentliche Repository ist unter der folgenden URL erreichbar: https://hub.docker.com/repository/docker/dvdrhr/discobats/tags [letzter Zugriff: 17.02.26]

Die Konfiguration der Remote Hosts wird als Teil der *Configuration* (vgl. Abschnitt 14.3.2.8 *Szenario*) innerhalb des Szenario-XML-Dokuments vorgenommen. Eine beispielhafte Verteilungskonfiguration ist in Listing 6 zu sehen. Um die Container auf den jeweiligen Remote-Hosts zu starten, müssen auf diesen Befehle ausgeführt werden können. Im Rahmen der prototypischen Umsetzung wurden Aspekte der Sicherheit und des Datenschutzes zunächst bewusst ausgeklammert, da diese kein direkter Teil der Zielsetzung dieser Arbeit sind und das angestrebte prototypische Ergebnis kein ohne Weiteres produktiv einsetzbares Softwareartefakt ist. Daher wurde zunächst auf unkompliziert umzusetzende, aber funktional ausreichende *Secure Shell*-(SSH-)Verbindung und eine Konfiguration der nötigen Zugangsdaten als Klartext zurückgegriffen.

Das Erzeugen des Docker-Images, auf Basis einer, auf den von einem Remote-Federate benötigten Funktionsumfang reduzierten, Variante der Simulationsanwendung und der von der Szenarioinstanz referenzierten Modellbibliothek, das Herstellen der SSH-Verbindung, das Übertragen der zum Start nötigen Szenarioinstanz-XML-Datei und das Aufrufen des Startbefehls für den Container geschieht automatisch während der Initialisierung der Simulationsanwendung, wenn die Szenarioinstanz eine Verteilungskonfiguration enthält. Die für diese Abläufe zuständigen Klassen sind verwaltend *SimulationManager* und ausführend *CustomClientBuilder* sowie *SSHService*.

Sollen Federates über mehrere lokale Netzwerke (LAN) verteilt ausgeführt werden, wenn also Federates z. B. das Internet als Kommunikationsmedium nutzen sollen oder müssen, dann muss zusätzlich noch die eigenständige Portico Komponente *WAN-Router*[96] laufen (d. h. gestartet und aktiv sein), bevor der Simulationslauf durch die Simulationsanwendung initialisiert wird. Die Netzwerke und Firewalls müssen dabei so konfiguriert sein, dass der konfigurierte Port des Hosts, auf dem der *WAN-Router* läuft, von allen Federates erreichbar ist. In jeder Umgebung (d. h. in jedem lokalen Netzwerk), in der Federates ausgeführt werden, muss zudem genau ein Federate die Rolle des lokalen Gateways übernehmen. Diese Rollenzuweisung wird durch die Simulationsanwendung automatisch vorgenommen. Die Federates innerhalb einer Umgebung kommunizieren in einem solchen Szenario, wie bei Einsatz von Portico üblich, mittels Multicast-Nachrichten. Das jeweilige Gateway-Federate baut zu Beginn zusätzlich eine Verbindung zum WAN-Router auf und leitet Nachrichten an den WAN-Router weiter, welcher wiederum bei Bedarf Nachrichten an alle weiteren verbundenen Gateway-Federates anderer Umgebungen weiterleitet (vgl. Abbildung 99). Ein direkt nutzbares Docker-Image, das eine Instanz des *WAN-Routers* startet und diesen auf dem Port 23114 des Host-Systems ansprechbar macht, ist aus dem Docker-Hub Repositorium des Umsetzungsprojekts abrufbar[97].

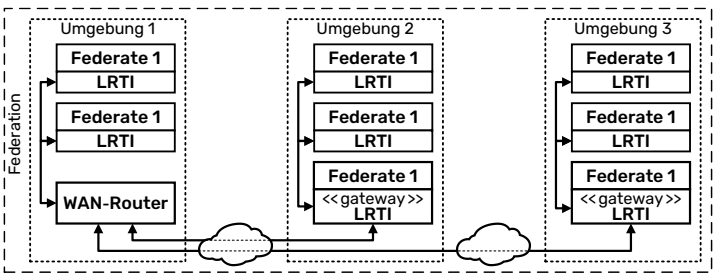


Abbildung 99: Portico WAN-Router als zentraler Kommunikationsknoten bei verteilter Ausführung

[96] The Portico Project, *Using Portico over a WAN*: `https://timpokorny.github.io/public/documentation/user/wan.html` [letzter Zugriff: 17.02.26]

[97] Docker Inc., Docker Hub, *Image Layer Details - dvdrhr/discobats:wan_router:* `https://hub.docker.com/repository/docker/dvdrhr/discobats/tags/wan_router/` [letzter Zugriff: 17.02.26]

16 ÜBERPRÜFUNG UND BEWERTUNG

Das entstandene prototypische Softwareartefakt in Form eines Frameworks zur Modellierung, Spezifikation und Durchführung verteilter Verkehrssimulationsläufe (vgl. Abschnitt 15) lässt sich nur dann belastbar bewerten, wenn neben der reinen Funktionsfähigkeit auch seine praktische Einsetzbarkeit und der Mehrwert des Einsatzes betrachtet werden. Daher folgt die Durchführung der Evaluation einer dreistufigen Methodik in Anlehnung an Oppermann et al. [2019] (vgl. Abbildung 100). Da der Untersuchungsgegenstand kein klassisches Interaktionssystem, sondern ein softwaretechnisches Framework ist, wird die angewandte Methodik als Übernahme der Evaluationslogik verstanden: Das Vorgehen wurde an die spezifischen Anforderungen verteilter Modellierungs- und Simulationsumgebungen angepasst und die Kriterien wurden aus den Zielen und Anforderungen dieser Arbeit abgeleitet und operationalisiert. Die Prüfung der Kriterien erfolgt anhand geeigneter Evidenzen: nachvollziehbare argumentative Schlussfolgerungen, demonstrative Testläufe und reproduzierbare Experimente. Im Zentrum der bewertenden Betrachtung steht die Frage der Erfüllung der für diese Arbeit gestellten Ziele durch den Prototyp und schlussfolgernd auch durch die erarbeiteten Lösungskonzepte. Zur Beantwortung dessen werden die durch Oppermann et al. identifizierten Nutzungsbeziehungen zwischen den drei Elementen des Aufgabenumfelds, *Computer*, *Benutzer* und *Aufgabe,* separat untersucht. Die Teilfragen, die es dabei zu beantworten gilt, sind, ob der Prototyp die formulierten lösungsorientierten Anforderungen erfüllt, wie gut er sich für die intendierten Einsatzkontexte eignet und welchen zusätzlichen Nutzen er darüber hinaus in der Lage ist, zu erbringen. Entsprechend verfolgt dieses Kapitel eine Evaluation, die sowohl einen klaren Abgleich mit den Anforderungen als auch eine anwendungsnahe Betrachtung der Nutzung ermöglicht.

16.1 UNTERSUCHUNG DER ANFORDERUNGSERFÜLLUNG DES PROTOTYPS

Im Rahmen der Untersuchung der Anforderungserfüllung wird der Abgleich des erstellten Prototyps mit den in Abschnitt 13 aufgestellten lösungsbezogenen Anforderungen vorgenommen. Die Untersuchung der Erfüllung der Anforderungen erfolgt dabei argumentativ. Für jede aufgestellte Anforderung wird nachfolgend begründet, durch welche Entwurfsentscheidung und/oder Eigenschaft des Prototyps diese erfüllt wird.

Die Anforderungen lassen sich in Bezug auf die prototypische Umsetzung jeweils einem von drei technischen Artefakten und den mit diesen verbundenen Aktivitäten des in dieser Arbeit betrachteten Ausschnitts des Prozesses simulativer Durchführung szenariobasierter Testläufe (vgl. Abbildung 35) zuordnen. Dies sind: Szenariomodelle und ihre Erstellung, Szenarioinstanzen und ihre Spezifikation sowie die Simulationsanwendung und die Nutzung dieser. Die zentrale Rolle dieser drei technischen Artefakte und ihre Verwendung zur Beantwortung der aufgestellten Forschungsfragen, lässt sich auch

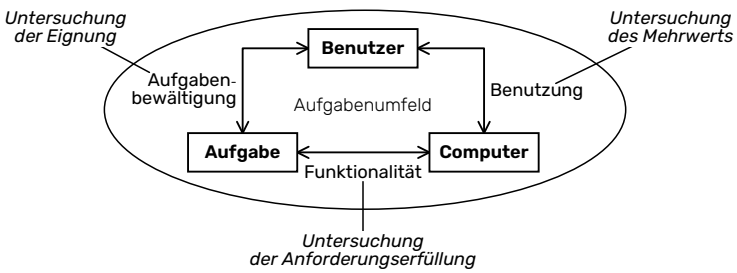


Abbildung 100: Elemente und Aktivitäten der mehrstufigen Evaluation (nach [Oppermann et al., 2019])

durch Bezugnahme zu den zu Beginn der konstruktiven Phase dieser Arbeit aufgestellten gestaltungsorientierten Teilziele erkennen: Die Teilziele lassen sich vollständig auf die genannten technischen Artefakte abbilden: Teilziel 1 adressiert die Modelle und Instanzen dieser, Teilziel 2 adressiert die flexibel einsetzbare Simulationsanwendung, Teilziel 3 adressiert die Modelle, die Spezifikation und die Simulationsanwendung, Teilziel 4 adressiert die Möglichkeiten im Rahmen der Spezifikation von Szenarioinstanzen, Teilziel 5 adressiert die Simulationsanwendung sowie die Modelle. Die nachfolgende Betrachtung der individuellen Anforderungen erfolgt systematisch anhand der drei genannten, eindeutig abgrenzbaren Bereiche.

16.1.1 ANFORDERUNGEN AN DIE MODELLIERUNG

Von den achtzehn in Abschnitt 13 aufgestellten lösungsbezogenen Anforderungen adressieren vier direkt das Szenariomodell oder den Modellierungsprozess. Die Konzept- und Entwurfsentscheidungen zur Begründung der Erfüllung dieser vier Anforderungen sind über mehrere Abschnitte dieser Arbeit verteilt zu finden. Den konzeptionellen Kern zur Erfüllung aller vier bilden dabei das Metamodell und die geschachtelte Modellhierarchie.

Die von der lösungsbezogenen Anforderung LA12 geforderte Abbildbarkeit von Verantwortungshierarchien wird durch die in Abschnitt 14.1.1 beschriebene Agentenarchitektur in Kombination mit der in Abschnitt 14.1.2 beschriebenen komponentenbasierten Komposition von Akteuren adressiert. Diese beiden übergreifenden Konzepte wurden in den Abschnitten 14.3.2.1, 14.3.2.2 und 14.3.2.5 strukturell konkretisiert sowie in den Abschnitten 14.3.3.4 und 14.3.3.6 funktional ausgestaltet. Durch die in Abschnitt 15 beschriebene Umsetzung, vordergründig der Ausführung der in eine Graphstruktur überführten Komponentenhierarchie und der Federate-zentralen Zustandshaltung, wurde die Anforderung erfolgreich adressiert.

Die konzeptionelle Grundlage für flexible Anpassungen und Erweiterungen der Modelle, wie sie von Anforderung LA14 verlangt wird, bildet die in Abschnitt 14.1.3 entworfene mehrdimensionale Modellierungshierarchie. Die Basis für die Umsetzung wurde in Form des Metamodells in Abschnitt 14.3.2 inhaltlich beschrieben. Die tatsächliche Verwendung von derart flexiblen Modellen wird durch die automatisierte Modellüberführung sowie -transformation ermöglicht (vgl. Abschnitte 14.3.3.1 und 14.3.3.2). Die Realisierung des transitiven Simulationsmodells im Rahmen der experimentellen Umsetzung beweist die Umsetzbarkeit dieser Lösungskonzepte und somit die Erfüllung der Anforderung.

Aufbauend auf der erwähnten mehrdimensionalen Modellierungshierarchie beschreibt das Konzept der *Szenariobibliothek* in Abschnitt 14.1.5.3 eine Möglichkeit zur Reduzierung der Nutzungskomplexität aus der Sicht der in Abschnitt 13.1 identifizierten Nutzerrolle des *Testers*. Die Realisierung dieses Konzeptes im Rahmen der experimentellen Umsetzung durch ein eigenes Projekt für jede ontologische Modellierungsebene gewährleistet die einfache Ableitbarkeit konkreter *Bibliotheken* auf Basis des allgemeinen Modells. Die Paketierung dieses Projekts als Java-Archiv (JAR), welches von einer Szenarioinstanz referenziert werden kann, vervollständigt die Erfüllung der Anforderung LA31.

Die letzte der vier Gruppen von Anforderungen adressiert nicht die Modelle selbst, sondern den Prozess der Erstellung dieser. Anforderung LA33 fordert, dass die Erstellung, Erweiterung und Anpassung von Modellen möglich sein müssen, ohne mit ablaufspezifischen Details der Simulationsanwendung in Berührung zu kommen. Adressiert wurde diese Anforderung durch die Verfolgung eines strengen und ganzheitlichen Ansatzes der Komplexitätskapselung, primär durch das Lösungskonzept des *Agen-*

tengerüsts (vgl. Abschnitt 14.1.5.2), die explizite Ausrichtung des Metamodells auf klare Schnittstellen zur Ausführung von Verhaltensimplementierungen (vgl. Abschnitt 14.3.2.5) sowie die zentrale Zustandshaltung (vgl. Abschnitt 14.3.3.4) und die automatische Modellinferenz (vgl. Abschnitt 14.3.3.5). Die funktionale Grundlage wurde als Konzept zur *impliziten Infrastruktur Anbindung* erarbeitet (vgl. Abschnitt 14.3.3.3) und in Form der vermittelnden *Model-Aware-RTI-Exchange* (maRTIx) Schicht realisiert. Die Integration dieser Schicht in die HLA-basierte Umsetzung erfüllt im Zusammenspiel mit der Umsetzung der ablaufsteuernden Federate-Container-Implementierungen (vgl. Abbildung 83 und Abbildung 95) letztlich die Anforderung durch Automatisierung und klare Schnittstellen.

16.1.2 ANFORDERUNGEN AN DIE SZENARIOSPEZIFIKATION

Sechs der in Abschnitt 13 formulierten lösungsbezogenen Anforderungen adressieren Szenarioinstanzen als technische Beschreibungen simulativ ausführbarer Szenarien oder die Erstellung dieser. Da das Metamodell und auf diesem basierende Modellbibliotheken die möglichen Inhalte für die Spezifikation und die Struktur von Szenarien vorgeben, überschneiden sich die erfüllenden Konzepte, Entwurfsentscheidungen und Umsetzungsbestandteile partiell mit den im vorigen Abschnitt genannten.

So fordert die lösungsbezogene Anforderung LA11, dass Szenarioinstanzen zu Testzwecken flexibel bzgl. des Detailgrads / der Auflösung zu erstellen sein müssen. Diese Anforderung ist auf Basis des einheitlichen Metamodells, das sowohl die Grundlage für die Strukturierung der Inhalte einer Szenarioinstanz als auch die Grundlage für die Interpretation durch die Simulationsanwendung darstellt, erfüllt. Die Möglichkeit, aufbauend auf dem geschaffenen Metamodell aufeinander aufbauende Modelle mit steigendem Detailgrad zu definieren, diese beliebig zu erweitern und anzupassen sowie Modellbibliotheken zu definieren und zu exportieren, in Verbindung mit der Möglichkeit, eben jene Modellbibliotheken als Basis für die Spezifikation einer Szenarioinstanz zu nutzen, beweist die Erfüllung dieser Anforderung in hinreichendem Umfang.

Auch die von der lösungsbezogenen Anforderung LA13 geforderte Erstellung von Szenarioinstanzen nach einem einheitlichen Vorgehen, unabhängig von der konkret referenzierten Modellbibliothek, wird durch die Kombination aus ganzheitlichem Metamodell und Modellierungshierarchie erfüllt: Das Metamodell gibt die Struktur vor, mit welcher eine Szenarioinstanz abgebildet werden muss (vgl. Abschnitt 14.3.2.8). Unabhängig von dieser Struktur können die Elemente aber im Rahmen der gegebenen Möglichkeiten beliebiger Ausprägungen sein und durch die Simulationsanwendung unabhängig von der konkreten Ausprägung interpretiert, instanziiert und ausgeführt werden (vgl. Abschnitt 14.3.3 und 15.2). Durch die Fähigkeit der Simulationsanwendung, auf Basis des Metamodells, beliebige Modelle nutzen zu können und entsprechend der Szenarioinstanz das transiente Simulationsmodell zu initialisieren, ist auch die Anforderung LA15 als erfüllt anzusehen. Handelt es sich um Zustandsvariablen des simulierten Elements, im Verlauf dieser Arbeit als *Property* definiert, so kann der Tester zudem den angegebenen Wert um eine (SI-)Einheit und ein (SI-)Präfix anreichern.

Die lösungsbezogene Anforderung LA16 fordert, dass sich Szenarioinstanzen, deren Form in der bisherigen Argumentation im Rahmen dieses Abschnitts stets nicht explizit erwähnt wurde, persistieren lassen. Während der Erarbeitung der Konzepte wurde dafür genauer festgelegt, dass eine serialisierte Beschreibung der Szenarioinstanz existieren soll. Diese kann im Sinne von *Unmarshalling* unter Verwendung der referenzierten Modellbibliothek in eine flüchtige objektorientierte Variante, das erwähnte transiente Simulationsmodell, überführt werden (vgl. u. a. Abschnitt 14.3.3). Sowohl das übergrei-

fende Konzept als auch der konkrete Entwurf lassen hierbei die Frage nach einem konkreten Format für die persistente Speicherung aber offen, da diese nicht relevant für die generelle Funktionalität ist. Im Rahmen der experimentellen Umsetzung wurde anschließend XML als standardisierte, textbasierte Auszeichnungssprache gewählt, um Szenarioinstanzen sowohl menschen- als auch maschinenlesbar in serialisierter Form persistierbar zu machen. Die prototypische Umsetzung der Simulationsanwendung setzt indes eine JAXB-Implementierung zur Realisierung der Überführung in das transiente Simulationsmodell ein. Die prototypische Umsetzung beweist somit die Erfüllung dieser Anforderung.

Szenarioinstanzen und ihre XML-Repräsentationen orientieren sich strukturell stark an einer im Ursprung natürlichsprachlichen und wenig technischen Beschreibung möglicher Szenarioinhalte (vgl. Abschnitt 7.2.4 und 14.1.2). Die aus dieser Beschreibung hervorgehende Zusammensetzungsstruktur wurde annähernd identisch in das Metamodell übernommen (vgl. Abschnitt 14.3.2.8). Somit ist die inhaltliche Beschreibung weitestgehend frei von technischen Details möglich. Für die Konfiguration des Simulationslaufs (vgl. Abbildung 80) sowie die Konfiguration der aktiven Anbindung von externen Testobjekten (vgl. nächster Absatz) ist hingegen technisches und semantisches Detailwissen nötig. Da diese aber in den üblichen Anwendungsfällen nur einen geringen Anteil an der Spezifikation von Szenarien haben, kann die Anforderung LA32 trotzdem als überwiegend erfüllt angesehen werden.

Die Anforderung LA41 adressiert als erste an dieser Stelle betrachtete Anforderung die Integration eines externen Testobjekts. Konkret fordert sie, dass die Beschreibung der Integration des Testobjekts bereits Teil der Szenarioinstanz und ihrer serialisierten Beschreibung sein muss. Das erarbeitete Konzept und die konkrete Ausgestaltung dessen in Form des Metamodells setzen dafür auf die *ProxyBehaviour* genannten funktionalen Platzhalter, die an die Stelle von zu simulierenden Verhaltensbeschreibungen treten. Auf Basis dessen, dass beide *Behaviour*-Varianten von derselben Superklasse erben, fügt sich die Integration externer Textobjekte nahtlos ein.

16.1.3 ANFORDERUNGEN AN DIE SIMULATIONSANWENDUNG

Acht der in Abschnitt 13 formulierten lösungsbezogenen Anforderungen richten sich an die Fähigkeiten der Simulationsanwendung oder der Nutzung dieser. Die Entwurfsentscheidungen sind somit hauptsächlich in den Abschnitten 14.3.3 *Simulation* und 14.3.4 *Verteilte Ausführung* verortet. Der Simulationsanwendung und den zugrundeliegenden Abläufen ist darüber hinaus ein Großteil des Abschnitts 15 bezüglich der experimentellen Umsetzung gewidmet.

Die lösungsbezogene Anforderung LA21 ist in Bezug auf die Simulationsanwendung wohl eine der gewichtigsten, denn sie basiert auf einem zentralen funktionalen Aspekt des übergreifenden Konzepts: der Initialisierung und Durchführung eines Simulationslaufs allein auf der Basis einer Szenarioinstanz und einer durch diese referenzierten Modellbibliothek. Entsprechende Konzept- und Entwurfsentscheidungen zur Realisierung dieser Anforderung wurden in den Abschnitten 14.1.4, 14.1.7 und 14.1.8 sowie in den Abschnitten 14.3.3 und 14.3.4 dargestellt. Über die bereits mehrfach erwähnte Modellüberführung hinaus ist die Simulationsanwendung gezielt in einer Art aufgebaut, die eine automatisierte Initialisierung aller nötigen Strukturen auf Basis einer Szenarioinstanz und des referenzierten Modells durchführt. Besonders relevant dafür sind die entworfenen Arten eines Federate-Containers als Umgebung für verschiedene Elemente, die unterschiedliche Funktionalitäten während eines Simulationslaufs erfüllen. Die prototypische Umsetzung hat gezeigt, dass die Summe dieser Vielzahl an Designentscheidungen die Anforderung erfolgreich erfüllt.

Die lösungsbezogene Anforderung LA22 überschneidet sich bezüglich der Lösungskonzepte mit der zuvor betrachteten Anforderung LA21: Das Metamodell erlaubt es Akteuren, eine beliebig tiefe und breite Komponentenhierarchie zuzuordnen (vgl. Abschnitt 14.3.2.2), welche durch die Simulationsanwendung in einen ausführbaren Zustand überführt werden muss. Das Konzept, das diese Anforderung direkt adressiert und erfolgreich realisiert, wurde in Abschnitt 14.3.3.6 *Komponentenbasierte Wirkstrukturen* ausführlich unter dem Namen *Behaviour Graph* vorgestellt.

Zum Zwecke der breiteren Einsatzbarkeit des vorgestellten Gesamtkonzepts fordern LA23 und LA24 sowohl die Abbildbarkeit von stochastisch beeinflusstem Verhalten agierender Modellelemente als auch von deterministischem Verhalten, welches bei jedem Simulationslauf die gleiche vorhersehbare Zustandstrajektorie erzeugt. Die erarbeiteten Konzepte bieten einen Rahmen, in welchem verschiedenste Modell- und Verhaltensausprägungen umsetzbar und simulativ ausführbar sind. Zu diesem Zweck sind Schnittstellen, Strukturen und Abläufe, mit denen der Modellentwickler in Kontakt kommt, bewusst so einfach gehalten worden, dass sie möglichst flexibel nutzbar sind. Für die Umsetzung von Verhalten besonders relevant sind dabei die *Zentrale Zustandshaltung*, über welche Verhaltensbeschreibungen Informationen erhalten, aktualisieren und miteinander teilen können, sowie die Schnittstelle zur Umsetzung von *Verhalten*, welche außer dem übergebenen Wert zur Beschreibung der seit dem letzten Simulationsschritt verstrichenen Zeit keinerlei einschränkende Vorgaben macht. Beide Anforderungen sind somit als erfüllt anzusehen, da der Modellentwickler frei in der konkreten Ausgestaltung der Verhaltensweisen ist.

Sowohl die Anforderung LA25 als auch die Anforderung LA26 adressieren die Verfügbarkeit der Zustandstrajektorie eines Simulationslaufs. Beide Anforderungen sind durch den Entwurf und die Umsetzung des *GlobalObserver*-Elements (vgl. Abschnitt 14.3.2.8) und der generischen *Property* zur Beschreibung von Zustandsvariablen von simulierten *Elementen* (vgl. Abschnitt 14.3.2.3) realisiert worden. Über das *GlobalObserver*-Element kann in einer Szenarioinstanz flexibel festgelegt werden, welche Werte welcher Elemente ausgegeben werden sollen. Als Teil der Modelle können, zusätzlich zu den vorhandenen, individuelle *OutputWriter* implementiert und innerhalb einer Szenarioinstanz einem *GlobalObserver* zugewiesen werden. Somit ist die Ausgabe der Zustandstrajektorie in einer Art flexibel, die es erlaubt, sowohl persistente Ausgaben für eine anschließende Auswertung als auch flüchtige Ausgaben für eine Beobachtung zur Laufzeit zu erzeugen, und beide Anforderungen können als erfüllt betrachtet werden.

Durch das an dieser Stelle bereits mehrfach angeführte, von der Simulationsanwendung selbstständig durchgeführte Unmarshalling und die eigenständige Initialisierung des Simulationslaufs sind keine durch den Tester manuell durchzuführenden Transformationen der Szenarioinstanzen oder -modelle notwendig. Die erfolgreiche experimentelle Umsetzung beweist somit, dass die erarbeiteten Konzepte und inhaltlichen Entwürfe die Erfüllung der Anforderung LA34 erfolgreich realisieren.

Die lösungsbezogene Anforderung LA51 fordert, aufbauend auf der Anforderung LA41, dass die Simulationsanwendung in der Lage ist, bidirektional mit allen in Tabelle 5 dargestellten möglichen Ausprägungen externer Testobjekte zu kommunizieren. Die zuvor identifizierten Ausprägungen konnten aus einer infrastrukturellen Sichtweise zusammenfassend auf zwei technische Ausprägungen reduziert werden. Das Konzept konzentrierte sich anschließend auf eine verteilt ausführbare Simulation (vgl. Abschnitt 14.1.7), um die Anbindung auch unter herausfordernden infrastrukturellen Bedingungen,

wie der Verortung des Testobjekts außerhalb des lokalen Netzwerks, durch den Einsatz von *Proxy*-Elementen zu ermöglichen. Umfangreich konkretisiert wurde dieser Ansatz im Abschnitt 14.3.4.3. Um auch semantische Interoperabilität zu schaffen, wurde zusätzlich ein dreistufiger Vermittlungs- und Übersetzungsprozess erarbeitet und durch die Einführung des *Dictionary* sowie des *Communication*- und des *FormatAdapter*s vervollständigt. Die erfolgreiche, flexibel auf ein konkretes Testobjekt anpassbare Ermöglichung syntaktischer und semantischer Interoperabilität durch den Prototyp bestätigt die Erfüllung dieser letzten Anforderung.

16.2 UNTERSUCHUNG DER EIGNUNG DES PROTOTYPS

Die Entwicklung eines Prototyps ist nur dann zielführend, wenn seine Qualität nicht ausschließlich über die Implementierung einzelner Funktionen, sondern über seine Wirkung im Anwendungskontext bewertet wird. Die Untersuchung der Eignung betrachtet daher, inwieweit der Prototyp die relevanten Aspekte typischer Einsatzszenarien im Kontext der Zielsetzung unterstützt. Dies sind etwa die Orchestrierung von Simulationsläufen, die Integration externer Komponenten, der Umgang mit verteilten Ressourcen sowie die Auswertbarkeit von Experimenten.

Die Kernkonzepte wurden zu diesem Zweck zunächst separat betrachtet und der Prototyp durch die Umsetzung kleinerer, isolierter, auf diese ausgerichteter Beispiele auf seine Eignung in Bezug auf den jeweiligen abgegrenzten Aufgabenbereich hin untersucht. Eine Herausforderung bei der ganzheitlichen Untersuchung der Eignung eines prototypisch umgesetzten Lösungskonzeptes in Form eines Software-Frameworks ist die Identifikation geeigneter Fallbeispiele. Da die Lösungskonzepte auf eine möglichst flexible Einsetzbarkeit hin ausgerichtet sind, ist die Anzahl theoretisch möglicher Anwendungsfälle groß und vielfältig. Wenn auch wünschenswert, ist die Erprobung aller Fälle aus dieser Menge kein realistisch umsetzbares Vorhaben. Um die Anwendbarkeit trotzdem in realen Anwendungsfällen zu demonstrieren, wurden zwei repräsentative Anwendungsfälle identifiziert, die reale externe Testobjekte und Softwaresysteme integrieren. Für diese werden in Abschnitt 16.2.2 jeweils ein Testfall entwickelt, umgesetzt und durchgeführt.

16.2.1 INDIVIDUELLE UNTERSUCHUNG DER KERNKONZEPTE

In Orientierung an den drei eingangs in Abschnitt 3 aufgestellten Teilforschungsfragen werden nachfolgend drei Funktions- und Eigenschaftsbereiche anhand beispielhafter Umsetzungen gezielt betrachtet: Flexibilität von Modellen und Ausführung (Abschnitt 16.2.1.1), Komplexitätskapselung (Abschnitt 16.2.1.2) und Interoperabilität (Abschnitt 16.2.1.3).

16.2.1.1 ERPROBUNG DER FLEXIBILITÄT VON MODELLEN

In Hinblick auf die erste Teilforschungsfrage *„Wie müssen Simulationsanwendungen und Szenariomodelle aufgebaut sein, damit sichergestellt werden kann, dass diese möglichst weitreichend zu Verifizierungs- und Validierungszwecken von maritimen Fahrsystemen eingesetzt werden können?“* wurden Szenarioinstanzen für drei beispielhafte Modellzwecke erstellt, die jeweils unterschiedliche Anforderungen an den Detailgrad und die Auflösung des zugrundeliegenden Modells haben. Dabei wurde ein Fokus darauf gelegt, auf früheren Modellebenen vererbend aufzubauen, um so gleichzeitig den praktischen Nutzen der angestrebten erhöhten Wiederverwendbarkeit zu untersuchen.

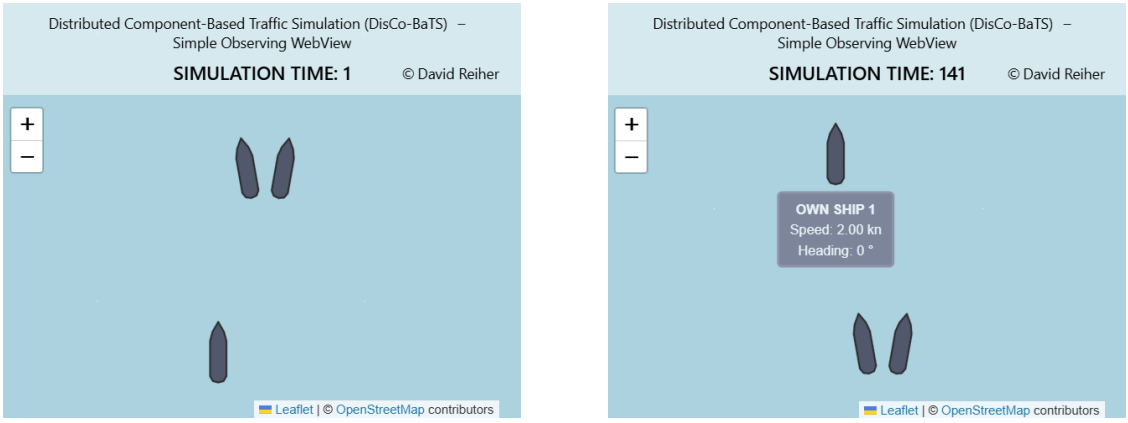


Abbildung 101: Screenshots von der Benutzeroberfläche der umgesetzten einfachen Webanwendung *WebView* zur Darstellung von durch den im Szenario konfigurierten *GlobalObserver* per *Websocket* gesendeten Daten während eines Simulationslaufs. Links: Zustand des Simulationslaufs auf Basis der Szenarioinstanz aus *Modellzweck 1* zum Zeitpunkt $t = 1$. Rechts: Zustand desselben Simulationslaufs zum Zeitpunkt $t = 141$.

Modellzweck 1

Der erste Modellzweck orientiert sich an dem Szenarioraum, der in [Austel et al., 2024] im Abschnitt *Case Study 2: Testing a vessel recognition system* vorgestellt wird. Die vorgestellten Szenarien sind auf einem abstrakten Niveau, repräsentieren bezüglich des Detailgrads jedoch eine Art von Szenarien, wie sie in wissenschaftlichen Veröffentlichungen häufig Anwendung finden. Eine auf den Modellzweck ausgerichtete Modellbibliothek wurde entsprechend umgesetzt[98].

Die Umsetzung der zugrundeliegenden Modellhierarchie zeigt eine bisher nicht erwähnte Möglichkeit bezüglich der Vererbungshierarchie: Die Modellbibliothek auf Ebene O_0 basiert direkt auf dem im Konzept als Teilmodell angedachten Paket D_1 der Ebene O_1 (vgl. Abschnitt 14.1.3). Die Anzahl der zur weiteren Aufteilung der Ebene O_1 Pakete ist keineswegs eine feste Größe. Vielmehr hängt die Anzahl ebenfalls vom Modellzweck, den konkreten Anforderungen und Bedürfnissen des Einsatzes sowie der Nutzungsumgebung ab. In diesem konkreten Fall mussten lediglich ein sehr abstraktes, nicht weiter spezifiziertes, Schiff abgebildet und ein *Task* zur Berechnung einer neuen Position auf Basis der aktuellen Position, der Geschwindigkeit und der aktuellen Rotation umgesetzt werden. Ein Detailgrad war somit ausreichend, welcher durch das Paket *de.uol.discobats.model.o1.d1.abstractmaritime* umgesetzt ist. Die exportierte Bibliothek definiert, basierend auf dem genannten Paket, das konkrete Element *UnspecifiedVessel*, um nicht benötigte *Properties* mit Standardwerten zu initialisieren, und das *SimpleStraightlineDrivingBehaviour*, welches der *ACTION*-Phase den o. g. Task hinzufügt. Die XML-Beschreibung der Szenarioinstanz, die die entstandene Modellbibliothek referenziert und eine konkrete Wertebelegung des o. g. Szenarioraums abbildet, ist in Anhang B vollständig dargestellt. Diese wurde als Eingabeparameter für die prototypische Simulationsanwendung genutzt, welche einen, der Beschreibung von Austel et al. entsprechenden, Simulationslauf durchführte (vgl. Abbildung 101).

Modellzweck 2

Aus Abbildung 8 geht hervor, dass ein Modell für die Erprobung von Fahraufgaben die Umgebung abbilden können muss. Je nach Detailgrad umfasst diese verschiedene Kombinationen und Ausprägungen von Konzepten wie Navigationsräumen, dynamischem Verkehr, Hindernissen und Verkehrszeichen. Wright [2020] betont im Kontext zusätzlich die Relevanz von *Aids to Navigation* (ATON), also

[98] GitHub – dvdrhr, *DisCo-BaTS, /models/o0-testcase/libraries/example-eval1*: `https://github.com/DisCo-BaTS/models/tree/main/o0-testcase/libraries/example-eval1` [letzter Zugriff: 15.05.26]

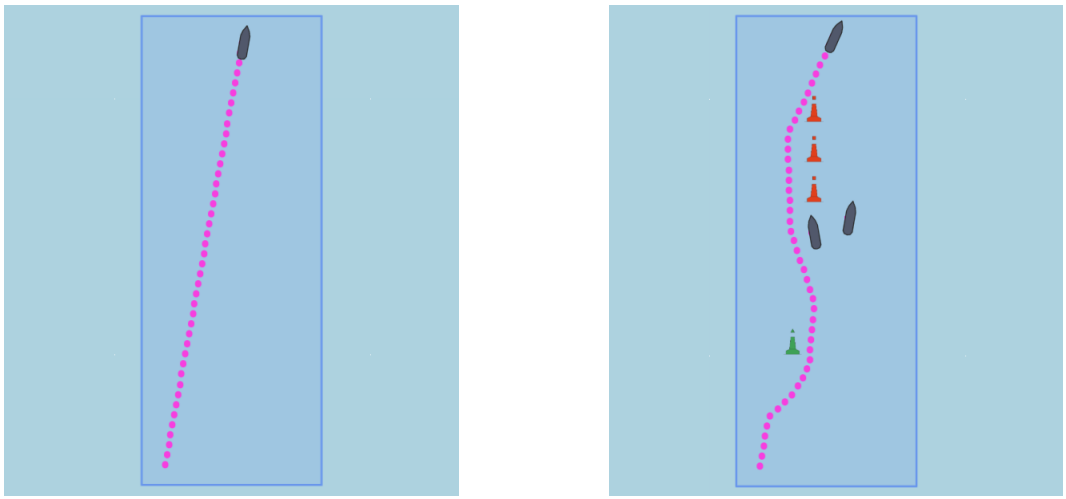


Abbildung 102: Screenshots der *WebView* während der für *Modellzweck 2* durchgeführten Simulationsläufe. Links: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne Hindernisse und andere Verkehrsteilnehmer zum Simulationszeitpunkt $t = 172$. Rechts: Zustand des Simulationslaufs auf Basis der um Hindernisse und weitere (passive) Verkehrsteilnehmer erweiterten Szenarioinstanz zum Simulationszeitpunkt $t = 184$. Das blaue Rechteck repräsentiert das Element *Waterbody*. Die roten und grünen Kegel repräsentieren *LateralMark*-Elemente des Typs *PORT_HAND* und *STARBOARD_HAND*. Die pinkfarbenen Kreise stellen die zurückgelegte Strecke des aktiven Schiffes dar.

allen Objekten, die zur Bestimmung der Position und zur Navigation genutzt werden können: Bojen, Leuchttürme, landseitige feste Seezeichen, Leuchtfeuer etc. In [Xiao et al., 2019] wird eine abstrakte dreiteilige Operationslogik autonomer Schiffe vorgestellt, welche die folgenden drei Schritte umfasst: *Origin-Destination route planning*, *Situational Awareness* und *Actions*.

Der zweite betrachtete Modellzweck kombiniert die beiden genannten Anforderungen beispielhaft: Es wurde ein Modell umgesetzt[99], das einfache Repräsentationen von Wasserkörpern als Begrenzung des befahrbaren Bereiches sowie eine Repräsentation des Lateralsystems durch zwei Arten von Bojen umfasst (vgl. Abbildung 102). Entsprechend der Phasen aus [Xiao et al., 2019] schreibt dieser konstruierte Modellzweck zudem ein entsprechend in mehrere Tasks aufgeteiltes Verhalten vor. Die in der ersten Phase relevante Route kann nach der Beschreibung von Xiao et al. extrinsisch zugeführt werden, weshalb das Modell um die Repräsentation von Routen in Form eines *Goals* angereichert wurde. Die zweite Phase nimmt Bojen sowie andere Schiffe wahr und dient als Basis für die dritte Phase, welche der gegebenen Route folgt, dabei bei Bedarf durch Kursänderungen anderen Schiffen ausweicht und versucht, innerhalb der Begrenzungen durch die Bojen des Lateralsystems zu bleiben.

Die Umsetzung der zusätzlichen Modellelemente erfolgte auf der Ebene O_1 (vgl. Abschnitt 14.1.3). Das umgesetzte Modellpaket setzt auf dem für *Modellzweck 1* erstellten Paket auf und erweitert die in diesem vorhandenen Klassen. Es stellt somit ein Paket D_2 im Sinne des Konzepts dar. Die Elemente dieses Pakets wurden analog zu *Modellzweck 1* anschließend in Form einer Modellbibliothek weiter konkretisiert. Die XML-Repräsentation der umgesetzten Szenarioinstanz ist in Anhang C dargestellt.

Modellzweck 3

Wright [2020] betont auch die Relevanz von Sensoren für automatisierte Schiffe. Aufgrund dessen, dass die Abbildung von systeminternen Wirkstrukturen durch hierarchische Anordnung von Komponenten eine zentrale Rolle in der Zielsetzung dieser Arbeit einnimmt, wurde der detaillierteste bei-

[99] GitHub – dvdrhr, *DisCo-BaTS, /models/o0-testcase/libraries/example-eval2*: `https://github.com/DisCo-BaTS/models/tree/main/o0-testcase/libraries/example-eval2` [letzter Zugriff: 15.05.26]

Listing 7: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs.

```
<component xsi:type="captain" timeStepSize="1">
  <executionGroup>2</executionGroup>
</component>
<component xsi:type="sensorDataProcessor" timeStepSize="1">
  <executionGroup>1</executionGroup>
  <components>
    <component xsi:type="sensorAIS" timeStepSize="1">
      <executionGroup>1</executionGroup>
    </component>
    <component xsi:type="sensorGPS" timeStepSize="1">
      <executionGroup>1</executionGroup>
    </component>
  </components>
</component>
```

spielhafte Modellzweck auf die Erprobung dieser Strukturen ausgerichtet. Konkret wurde das bestehende Modell aus dem vorigen Teilabschnitt *Modellzweck 2* um zwei Component-Elemente erweitert, die empfangende Sensoren repräsentieren: *GPS-Sensor* und *AIS-Sensor*. Diese nutzen den zentralen *State* bzw. die zentrale *Environment* des Akteurs, erzeugen auf Basis vorhandener Simulationsdaten NMEA 0183-konforme *Global Positioning System Fix Data*-(GGA-)Datensätze bzw. Bitrepräsentationen von *Automatic Identification System*-(AIS-)Nachrichten vom *Typ 1* und schreiben die erzeugten Daten in den zentralen *State.* Zusätzlich wurde eine datenvorverarbeitende Komponente integriert, welche verfügbare GPS- und AIS-Daten bei jeder Ausführung liest, wieder dekodiert und auf die Menge von AIS-Positionen reduziert, die in einem definierten Bereich vor der eigenen GPS-Position liegen. Die Komponente *Captain* bestimmt anschließend anhand dieser Positionen bei Bedarf einen ausweichenden Kurs des Schiffes. Das zu resultierende Verhalten ähnelt somit dem aus *Modellzweck 2.*

Die beschriebene Wirkstruktur wurde, wie in Abbildung 103 dargestellt, innerhalb eines weiteren Pakets D_3 umgesetzt[100], welches dem Konzept entsprechend ebenfalls auf Ebene O_1 einzuordnen ist und von dem Paket des vorigen Abschnitts D_2 erbt und die Elemente entsprechend erweitert. Die Spezifikation der beschriebenen Wirkstruktur innerhalb der XML-Repräsentation anhand von Schachtelung und Priorisierung durch die Angabe der *executionGroup* ist in Listing 7 abgebildet. Die vollständige XML-Repräsentation der beispielhaft erstellten Szenarioinstanz ist in Anhang D zu finden. Die simulative Ausführung letztgenannter sowie einer Variante ohne Hindernisse zeigte das erwartete Verhalten (vgl. Abbildung 104) und beweist somit die Eignung des Konzepts zur flexiblen Modellierung interner komponentenbasierter Wirkstrukturen von Verkehrsteilnehmern.

16.2.1.2 ERPROBUNG DER KOMPLEXITÄTSKAPSELUNG

Die zweite Teilforschungsfrage adressiert die Kritik an szenariobasierten simulativen Testprozessen: Der Prozess der Erstellung von Szenarien, der Umsetzung geeigneter Modelle und der anschließenden

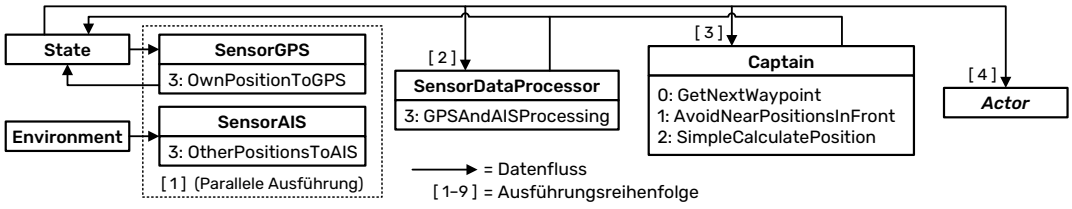


Abbildung 103: Komponenten der umgesetzten Wirkstruktur. Auflistend dargestellt sind pro Komponente die individuelle *BehaviourTask*-Konkretisierungen, die dem Verhalten dieser Komponente zugeordnet sind.

[100] GitHub – dvdrhr, *DisCo-BaTS, /models/o0-testcase/libraries/example-eval3*: https://github.com/DisCo-BaTS/models/tree/main/o0-testcase/libraries/example-eval3 [letzter Zugriff: 15.05.26]

Abbildung 104: Screenshots der *WebView* während der für *Modellzweck 3* durchgeführten Simulationsläufe. Links: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne Hindernisse und andere Verkehrsteilnehmer zum Simulationszeitpunkt $t = 72$. Rechts: Zustand des Simulationslaufs auf Basis der um (passive) Verkehrsteilnehmer erweiterten Szenarioinstanz zum Simulationszeitpunkt $t = 72$.

Durchführung eines Simulationslaufs sei zu komplex und zu zeitaufwendig. Als einer der Gründe dafür wurde im frühen Verlauf dieser Arbeit die Notwendigkeit von detaillierten Kenntnissen zu Strukturen und Abläufen der Modelle und Simulationsanwendungen identifiziert. Entsprechend wurde das Konzept gestaltet, entworfen und prototypisch umgesetzt, dass ein möglichst umfangreicher Teil der Komplexität von potenziellen Anwendern durch Kapselung ferngehalten wird (vgl. u. a. Abschnitt 14.1.5). Überdies wurden in Abschnitt 13.1 klar voneinander abgegrenzte Nutzergruppen eines szenariobasierten simulationsgestützten Testprozesses identifiziert und definiert.

Dass der entstandene Prototyp die gesetzten Ziele zur Reduzierung der Menge nötigen aufgabenfremden Wissens erfüllt, kann anhand der umgesetzten Beispiele des vorigen Abschnitts argumentativ geschlussfolgert werden. An dieser Stelle bedarf es somit keiner eigenen beispielhaften Anwendungsfälle.

Im vorangegangenen Abschnitt zur *Erprobung der Flexibilität von Modellen* wurden Modelle, Bibliotheken und Szenarioinstanzen unterschiedlichen Detailgrads, ausgerichtet auf drei beispielhafte Modellzwecke, erstellt. Die Szenarioinstanzen dienten anschließend als Eingabe für die prototypische Simulationsanwendung zum Zwecke der Durchführung jeweils eines Simulationslaufs. Die erfolgreiche Durchführung dieser Simulationsläufe und die erwartungsgemäße Entwicklung der simulierten Elemente können anhand von Abbildung 101, Abbildung 102 und Abbildung 104 nachvollzogen werden.

Die im Rahmen der erwähnten demonstrativen Beispiele durchgeführten Aufgaben lassen sich zwei der identifizierten Rollen zuordnen: Die Erstellung der Modelle, d. h. der Inhalte der Ebenen O_2 und O_1 der als Beitrag zur Lösung konzipierten Modellhierarchie, und der Modellbibliotheken auf der Ebene O_0 sind dem Aufgabenbereich des *Modellentwicklers* zuzuordnen. In Abschnitt 13.1 wurde festgelegt, dass ein *Modellentwickler* zwar Programmierkenntnisse sowie Kenntnisse über die Schnittstelle des Metamodells und der auf diesem basierenden Simulationsanwendung besitzen muss, aber keinen direkten Kontakt mit internen Mechaniken des Simulationsablaufs selbst hat. Zentrale Konzeptbestandteile, um diese Trennung von Simulationsanwendung und Modellen zu schaffen, sind das entworfene Metamodell, die Realisierung von Verkehrsteilnehmern als Agenten sowie das Agentengerüst und seine geringe Anzahl zentraler Schnittstellen. Die beiden letzten Punkte werden konkret primär durch aufgabenspezifische Federate-Instanzen als Laufzeitcontainer für die Modellelemente (vgl. Abschnitt 14.3.3.1) sowie die modellbewusste RTI-Austauschschicht (vgl. Abschnitt 14.3.3.3) realisiert.

Listing 8: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs

```
@ScenarioSpecificationItem
public class SeagoingVessel extends Vessel {

    @ExchangeableProperty
    private ImmutableProperty<String> imo;

    @ExchangeableProperty
    protected ImmutableProperty<String> mmsi;

    public SeagoingVessel() { super(); }
}
```

Listing 9: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs

```
public class OtherPositionsToAISTask extends BehaviourTask {
    public static final String STATE_TOPIC_AIS_BIT = "data.sensors.position.ais.bit";
    public OtherPositionsToAISTask(SimulationBehaviour behaviour) { super(behaviour); }
    @Override
    public void execute(int timePassed) {
        List<SeagoingVessel> otherVessels = environment.get(SeagoingVessel.class).stream().map(
                                                element -> (SeagoingVessel) element
                                             ).toList();
        List<String> aisPositions = otherVessels.stream().map(vessel -> {
                                            Position vesselPos = vessel.getPosition().getValue();
                                             return AISData.encode(vesselPos.getLatitude().getValue(),
                                                                   vesselPos.getLongitude().getValue(),
                                                                   vessel.getMmsi().getValue());
                                    }).toList();
        state.put(STATE_TOPIC_AIS_BIT, aisPositions);
    }
    @Override
    protected void setup() {  // nothing to do  }
}
```

Listing 8 und Listing 9 zeigen beispielhafte Ausschnitte aus den im Rahmen der Untersuchung von *Modellzweck 3* im vorigen Abschnitt umgesetzten Modellen. Listing 8 zeigt eine Konkretisierung der abstrakten Klasse *DynamicReactor*, die die strukturelle Ausprägung *DynamicPhysicalElement* und *Reactor* kombiniert. Die gezeigte Klasse *SeagoingVessel* erbt dabei von dem Element *Vessel* des Modellpakets, das im Rahmen der Untersuchung von *Modellzweck 2* entstanden ist. Die Vererbung, die Annotationen sowie das Anreichern der Attribute durch Verpacken dieser in eine der beiden generischen *Property*-Ausprägungen *ImmutableProperty<T>* und *MutableProperty<T>* stellen die strukturellen Modellierungsschnittstellen dar. Listung 9 lässt als Gegenstück dazu die ablaufbezogenen Schnittstellen erkennen: Abgebildet ist die vollständige Klasse *OtherPositionToAISTask*, die das Verhalten des beispielhaft umgesetzten *AISSensor* definiert. Die von Verhaltensbeschreibungen zu nutzende Schnittstelle ist dabei dreigeteilt: (1) Struktur: das Ableiten der Klasse von *BehaviourTask* sowie das Einbinden in eine der Phasen eines konkreten *Behaviour*; (2) Ablauf: die Umsetzung eines *Tasks* muss die Methoden *execute* und *setup* überschreiben. Die erstgenannte wird bei jedem Fortschreiten in der Simulationszeit aufgerufen und die zweitgenannte einmalig während der Initialisierung des Federates; (3) Datenaustausch: Die zentralen Datenhaltungsobjekte *State* und *Environment* stehen automatisch innerhalb jeder *Task*-Implementierung zur Verfügung und können über die Felder *state* und *environment* genutzt werden. Die durch diese Gruppe an strukturellen und funktionalen Schnittstellen geschaffene klare Grenze wird anhand der umgesetzten Beispiele und der beispielhaften Ausschnitte hinreichend bewiesen, um das Ziel der Kapselung der Komplexität für die Rolle des Modellentwicklers als erreicht anzusehen.

Anhand der im vorigen Abschnitt spezifizierten Szenarioinstanzen im XML-Format (Anhänge B, C und D) ist zu erkennen, dass für die Ausübung der Rolle als *Tester* lediglich wenige Kenntnisse über

die tiefere Modellstruktur notwendig sind. Funktionale Details sind auf dieser Modellierungs- bzw. Spezifikationsebene nicht vorhanden, da die Simulationsanwendung in der Lage ist, beliebige Szenariospezifikationen auf Basis beliebiger Modellbibliotheken ohne weitere manuelle Modelltransformationen/-anreicherungen oder notwendige Konfigurationen auszuführen. Das Erreichen des angestrebten Grads an Komplexitätskapselung für die Rolle des *Testers* ist somit ebenfalls hinreichend erkennbar.

16.2.1.3 ERPROBUNG DER INTEROPERABILITÄT

Die simulative Ausführung von maritimen Verkehrsszenarien geschieht im ausführlich aufgearbeiteten Kontext dieser Arbeit nicht zum Selbstzweck. Stattdessen dient sie als Werkzeug für die Erprobung und den Beweis der korrekten und sicheren Funktionsweise von softwarebasierten Systemen, Teilsystemen und Komponenten automatisierter Wasserfahrzeuge (vgl. Teil I und II). Um dieser Aufgabe gerecht zu werden, ist eine der Grundvoraussetzungen, dass das Testobjekt aktiv in den Simulationslauf integriert ist (vgl. Abschnitt 7.3.4). Teilforschungsfrage 3 adressiert diese Herausforderung daher ebenfalls direkt: „*Wie kann ermöglicht werden, dass heterogene zu testende Systeme am Simulationslauf teilnehmen können, ohne dass Änderungen oder Anpassungen an der Simulationsanwendung oder am zu testenden System selbst vorgenommen werden müssen?*“ Die übergreifenden Teilkonzepte zur Ermöglichung einer solchen generischen Interoperabilität wurden in den Abschnitten 14.1.6 und 14.1.7 dargestellt. Anschließend wurden in den Abschnitten 14.3.2.5 und 14.3.4 die entsprechende inhaltliche Ausgestaltung genauer betrachtet. Im Abschnitt 15.3 wurden zudem umsetzungsrelevante Details, wie die Wahl von Docker als konkrete Technologie und die Konfiguration der Verteilung innerhalb einer Szenarioinstanz, begründet dargestellt. Eine solche Konfiguration ist in Listing 10 beispielhaft dargestellt.

Zur Erprobung der Interoperabilität entsprechend der Zielsetzung wurde ein beispielhaftes Java-Programm entworfen und umgesetzt[101], welches in der Lage ist, Daten mittels UDP-Kommunikation zu empfangen und zu senden. Dabei kann es so konfiguriert werden, dass die Daten entweder im JSON-Format oder im XML-Format übertragen werden. Auch die inhaltliche Struktur der erwarteten und gelieferten Daten kann variiert werden. Zusätzlich kann die Frequenz, in welcher Daten gesendet

Listing 10: Ausschnitt der Konfiguration eines *ProxyBehaviours* in der XML-Repräsentation einer Szenarioinstanz

```
<behaviour xsi:type="proxyBehaviour">
  <mode>SYNC</mode>
  <inputCommunicationAdapter xsi:type="exampleUDPCommunicationAdapter">
    <localInPort>1339</localInPort>
  </inputCommunicationAdapter>
  <inputFormatAdapter xsi:type="exampleXMLFormatAdapter"/>
  <dictionary>
    <externalValues>
      <value>
        <name>speed</name>
        <path>vessel.characteristics[type=DynamicObjectCharacteristic].linearVelocity.value.y</path>
        <type>double</type>
        <unit>SPEED.METERS_PER_SECOND</unit>
        <prefix>NONE</prefix>
      </value>
    </externalValues>
    <input>
      <translation>
        <internalPath>self.main.speed</internalPath>
        <externalValue>speed</externalValue>
      </translation>
    </input>
  </dictionary>
</behaviour>
```

[101] GitHub – dvdrhr, *DisCo-BaTS, testunits*: https://github.com/DisCo-BaTS/testunits [letzter Zugriff: 15.05.26]

Abbildung 105: Screenshots der *WebView* während zwei zur Erprobung der Interoperabilität durchgeführten Simulationsläufe. Oben: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne integrierte externe Testobjekte zum Simulationszeitpunkt *t=600*. Unten: Zustand eines Simulationslaufs auf Basis der um die Integration eines externen Testobjekts, welches den Wert der Geschwindigkeit linear erhöht, erweiterten Szenarioinstanz zum Simulationszeitpunkt *t=450*.

werden, frei durch Startparameter angepasst werden. Erwartet und gesendet werden Daten, die ein Schiff repräsentieren. Eingehende und ausgehende Nachrichten sind dabei unterschiedlich umfangreich und weisen eine unterschiedliche Struktur auf. Eine Besonderheit ist, dass das beispielhafte Testobjektprogramm die Geschwindigkeitsangabe als Vielfaches der Lichtgeschwindigkeit erwartet, statt in Knoten oder Metern pro Sekunde. Das Programm erhöht vor jeder Übertragung ausgehender Daten den Wert der Geschwindigkeit des von den empfangenen Daten beschriebenen Schiffes. Die Szenariospezifikation der konkreten Testläufe wurde so gestaltet, dass die Simulationsanwendung diesen Geschwindigkeitswert nach einer automatisierten Übersetzung der *Unit* als neue Geschwindigkeit des Schiffes übernimmt, welchem die *ProxyBehaviour*-Komponente zugewiesen ist (vgl. Abbildung 105).

Für die Ermöglichung der bidirektionalen Kommunikation wurden im Sinne des erarbeiteten Konzepts die Klassen *ExampleXMLFormatAdapter, ExampleJSONFormatAdapter* und *ExampleUDPCommunicationAdapter* umgesetzt und verwendet. Es wurden mehrere Testläufe durchgeführt und die Konfiguration des Testobjekts wurde dabei bzgl. des Formats, der Frequenz und der maximalen Länge einer zufälligen Wartezeit vor jedem Senden zur Simulierung von Netzwerkverzögerungen jeweils variiert. Die in allen Varianten erfolgreiche Teilnahme des beispielhaften Testobjekts am Simulationslauf beweist somit die Fähigkeit des Prototyps, flexible semantische Interoperabilität zu ermöglichen.

Zur Erprobung der syntaktischen Interoperabilität wurde das beschriebene Szenario bezüglich der Verteilungskonfiguration variiert. Die drei unterschiedlichen Verteilungen der Anwendungsbestandteile auf Computersysteme (Hosts) und Netzwerke, die umgesetzt und erprobt wurden, sind in Abbildung 106 dargestellt. Weil auch die infrastrukturellen Variationen erfolgreich umgesetzt werden konnten und ein Simulationslauf entsprechend der Szenarioinstanz über alle Verteilungen hinweg möglich war, ist auch die syntaktische Interoperabilität hinreichend bewiesen.

16.2.2 GANZHEITLICHE UNTERSUCHUNG ANHAND EINER FALLSTUDIE

Um den Funktionsumfang möglichst vollständig abzudecken und die Übertragbarkeit der Ergebnisse der bisher eher konstruierten Untersuchungen auf reale Anwendungsfälle einschätzen zu können, erfolgte die ganzheitliche Untersuchung anhand einer repräsentativen Fallstudie. Die Fallstudie integriert ein existierendes softwarebasiertes Testobjekt. Die erstellten Testfälle bilden dabei die Bedarfe eines Einsatzes im Rahmen eines realen Entwicklungs- oder Forschungsvorhabens ab. So wird eine realitätsnahe Aussage über die Eignung der Lösungskonzepte und des auf diesen basierenden Prototyps

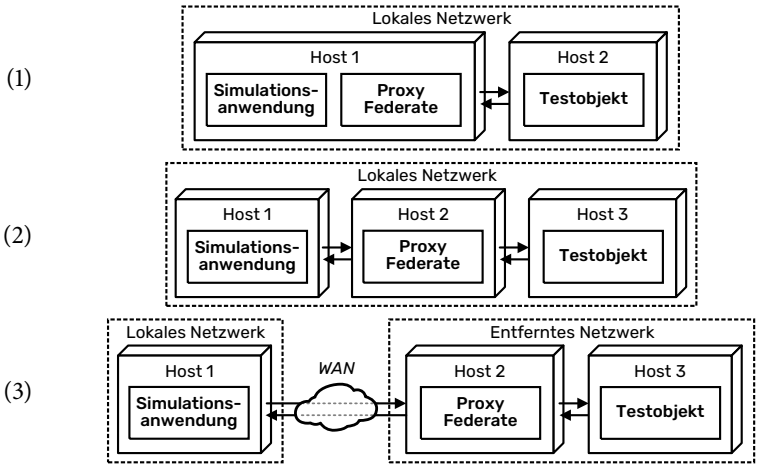


Abbildung 106: Zum Zweck der Untersuchung der Interoperabilität erprobte Verteilungsvarianten

ermöglicht. Nachfolgend wird kurz der betrachtete Fall vorgestellt, der Versuchsaufbau dargestellt, die Umsetzung des Testfalls umrissen sowie die Durchführung und die Ergebnisse besprochen.

Als Fall wurde die simulative Erprobung eines softwarebasierten maritimen Assistenzsystems ausgewählt, das, unabhängig von dieser Arbeit, im Rahmen eines Forschungsprojektes entstanden ist und im Rahmen dessen partiell simulativ erprobt wurde. Das *Maritime Traffic Alert and Collision Avoidance System* (MTCAS) ist ein Assistenzsystem zur proaktiven Kollisionsvermeidung. Zentraler Bestandteil ist ein eigens entwickelter Ansatz zur Berechnung des *Closest Point of Approach* (CPA), der eine kontextsensitive Vorhersage des Schiffsverhaltens einbezieht. Zur Berechnung der Vorhersage werden Informationen über typische Schiffsbewegungen, Tiefenprofile und Routeninformationen verwendet. Als Resultat unterstützt MTCAS Schiffsführer bei kritischen Begegnungen mit anderen Schiffen, indem proaktiv Ausweichmanöver verhandelt werden und eine Handlungsempfehlung zur Auflösung der Situation bereitgestellt wird. [Steidel & Hahn, 2019]

Im Rahmen der Evaluation von MTCAS wurden die entwickelten Kernfunktionalitäten in Form einer Verhaltenskomponente in den virtuellen Teil der *e-Maritime Integrated Reference Platform* (eMIR)[102] integriert [Steidel & Hahn, 2019]. Diese diente der durchgeführten Fallstudie als Untersuchungsgegenstand. Das zu erreichende Ziel war es somit, einen Testfall in Form einer Szenarioinstanz im Sinne des erarbeiteten Konzepts zu erstellen und die MTCAS-Komponente als Teil von eMIR in dieses zu integrieren. Die erstellte Szenarioinstanz musste anschließend als Basis eines Simulationslaufs dienen können, an welchem die MTCAS-Komponente aktiv teilnehmen konnte (vgl. Abbildung 107).

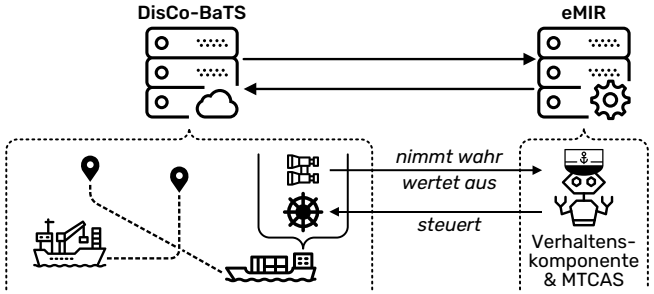


Abbildung 107: Fallstudie – Erprobung einer nach MTCAS-Vorschlägen ausweichenden Verhaltensimplementierung

[102] Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR), *eMIR – e-Maritime Integrated Reference Platform*: `https://www.dlr.de/de/se/forschung-transfer/forschungsinfrastruktur/emir` [letzter Zugriff: 17.02.26]

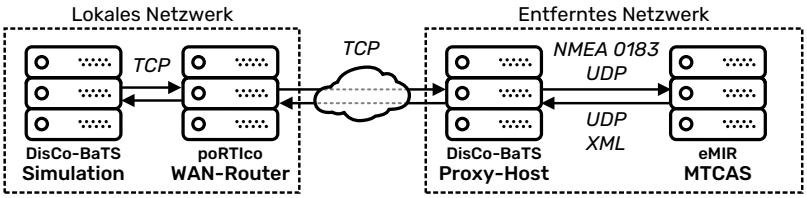


Abbildung 108: Fallstudie – Anbindung der MTCAS-Implementierung der *eMIR*-Plattform als externes Testobjekt

Der Versuchsaufbau wurde entsprechend der in Abschnitt 14.1.7 beschriebenen Herausforderung, der möglichen Notwendigkeit, logisch und geografisch entfernte Testobjekte in die simulative Ausführung eines Testfalls einzubinden, wie in Abbildung 108 dargestellt, verteilt gestaltet. Dabei kommen innerhalb eines Netzwerkes UDP und zwischen diesen TCP als Kommunikationsprotokolle zum Einsatz.

Es wurde ein beispielhaftes Modell umgesetzt[103], welches alle Schichten und Pakete des vorgeschlagenen Konzepts realisiert. Abbildung 109 zeigt einen Ausschnitt des umgesetzten Modells und ordnet die abgebildeten Modellelemente den Schichten der in Abschnitt 14.1.3 erarbeiteten Modellhierarchie vor.

Basierend auf diesem Modell wurde anschließend eine Modellbibliothek definiert, welche u. a. eine größtenteils vorparametrierte Variante des Elements *ContainerShip* zur vereinfachten Anwendung unter dem Namen *ExampleContainerShip* zur Verfügung stellt. In diesem Schritt wurden ebenfalls die für die bidirektionale Kommunikation mit der MTCAS-Komponente nötigen *Adapter* erstellt. In der verwendeten Konfiguration erwartet die MTCAS-Implementierung, genauer gesagt die entsprechende eMIR-Komponente, Eingabedaten in Form von NMEA 0183-konformen *Global Positioning System Fix Data* (GGA)-Nachrichten. Der Inhalt der GGA-Nachrichten muss dabei die Position des als *Targetship* bezeichneten Schiffes beschreiben, auf welches MTCAS reagieren soll. Ein entsprechender *FormatAdapter* wurde u. a. unter Einsatz der offenen Bibliothek *Java Marine API*[104] erstellt. NMEA 0183 ist ein im maritimen Bereich weitverbreiteter Standard, der u. a. in der Kommunikation zwischen Navigationsgeräten auf Schiffen zum Einsatz kommt [Wright, 2020]. Die Ermöglichung einer entsprechenden Datenübertragung im Rahmen der Fallstudie zeigt somit die praxisnahe Einsetzbarkeit der präsentierten Lösungskonzepte und des erstellten Prototyps.

Die angebundene eMIR-Komponente sendet in der genutzten Konfiguration zusätzlich aktuelle Zustandsgrößen des vom integrierten MTCAS-Verhalten gesteuerten Schiffes, wie die Position, die Aus-

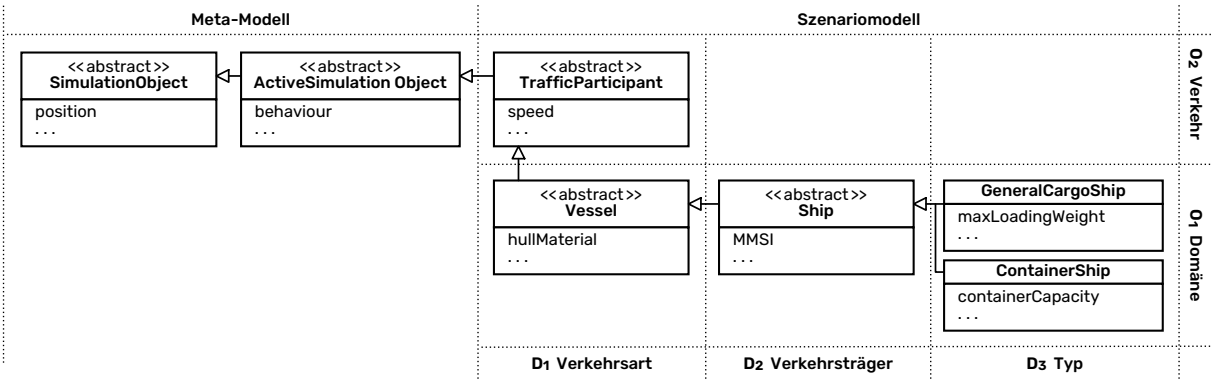


Abbildung 109: Ausschnitt des für Modells die Durchführung der Fallstudie umgesetzten Modells

[103] GitHub – dvdrhr, *DisCo-BaTS, /models/o1-domain/d3-type/transport*: `https://github.com/DisCo-BaTS/models/tree/main/o1-domain/d3-type/transport` [letzter Zugriff: 15.05.26]

[104] GitHub – Kimmo Tuukkanen, *Java Marine API*: https://github.com/ktuukkan/marine-api [letzter Zugriff: 17.02.26]

richtung und die Geschwindigkeit im XML-Format. Zum Zwecke der Ermöglichung einer zentralen Beobachtung und Auswertung der simulativen Testläufe wurde ein entsprechender zweiter *FormatAdapter* für die Überführung der eingehenden Daten in die anwendungsintern genutzte Struktur umgesetzt. Sowohl die eingehende als auch die ausgehende Datenübertragung der eMIR-Komponente erfolgt in der genutzten Konfiguration mittels UDP als Kommunikationsprotokoll. Ein entsprechender *CommunicationAdapter* wurde als abschließender Teil der Modellbibliothek ebenfalls erstellt.

Die umgesetzte Modellbibliothek[105] wurde genutzt, um einen Testfall abzubilden, der dem von Steidel & Hahn beschriebenen Typ der *1-Ship-Combination* entspricht. In dieser Kombination ist jeweils ein Schiff mit einer MTCAS-Implementierung und eines ohne Teil einer aufzulösenden potenziell kritischen Situation. Die konkrete Szenarioinstanz spezifiziert zwei reaktive Akteure vom zuvor beschriebenen Typ *exampleContainerShip*. Für erstgenanntes werden lediglich konkrete Wertebelegungen für die Attribute *id* und *name* festgelegt. Die restlichen Werte werden von der externen eMIR-Komponente geliefert, deren Anbindung über die Spezifikation eines *ProxyBehaviour* geschieht. Diesem wurden entsprechend Referenzen zu den zuvor beschriebenen Adaptern *exampleUDPCommunicationAdapter* und *exampleEMir-XMLFormatAdapter* zugewiesen.

Eine für die semantische Interoperabilität zentrale Konfiguration des *Dictionary*, wie sie beispielhaft in Listing 10 dargestellt ist, wurde für die Attribute *speed, position.latitude, position.longitude, rotation, course, mmsi, callsign* und *imo* vorgenommen. Die Spezifikation des zweiten Akteurs umfasst hingegen bereits konkrete Wertebelegungen für die Attribute *id, name, position.longitude, position.latitude, speed, rotation* sowie *course* und umfasst eine Referenz auf die interne Verhaltensbeschreibung vom Typ *simpleFollowRouteBehaviour.*

Eine entsprechende Route, bestehend aus einem einzelnen Wegpunkt, ist durch ein *RouteGoal* als Subtyp von *Goal* in die Szenarioinstanz integriert. Zur Übertragung der aktuellen Positionsdaten dieses Akteurs, welcher das *targetship* darstellt, ist diesem eine Komponente zugeordnet, welche wiederum eine Referenz auf ein weiteres *ProxyBehaviour* hält. Dieses hat entsprechend der Anforderung der externen MTCAS-Komponente Instanzen des *exampleUDPCommunicationAdapter* und des *exampleNMEA0183FormatAdapter* zugeordnet. Die Konfiguration des *Dictionary* beschreibt hier die Übersetzung der Werte der Attribute *position.latitude* und *position.longitude.*

Da beide Akteure mit der eMIR-Plattform kommunizieren, wurde die Verteilungskonfiguration mittels des *Distribution* Elements (vgl. Abschnitt 14.3.2.8) wie in Abbildung 110 dargestellt umgesetzt. Die vollständige XML-Repräsentation der erstellten Szenarioinstanz ist in Anhang F zu finden.

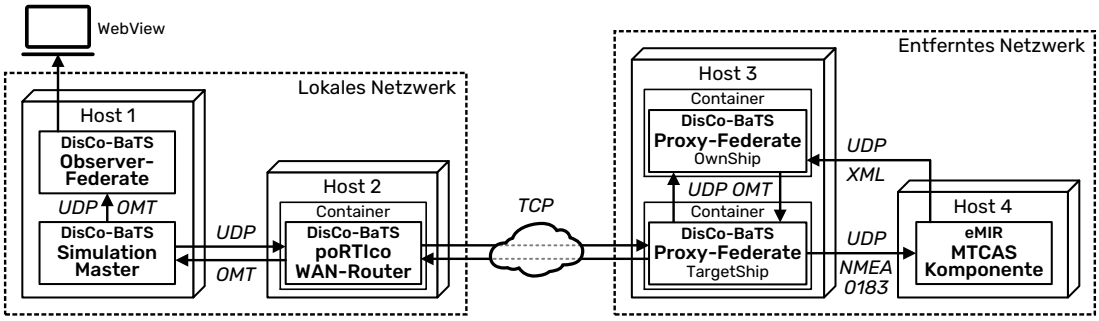


Abbildung 110: Detaillierter Versuchsaufbau der Fallstudie

[105] GitHub – dvdrhr, *DisCo-BaTS, /models/o0-testcase/libraries/example*: `https://github.com/DisCo-BaTS/models/tree/main/o0-testcase/libraries/example` [letzter Zugriff: 15.05.26]

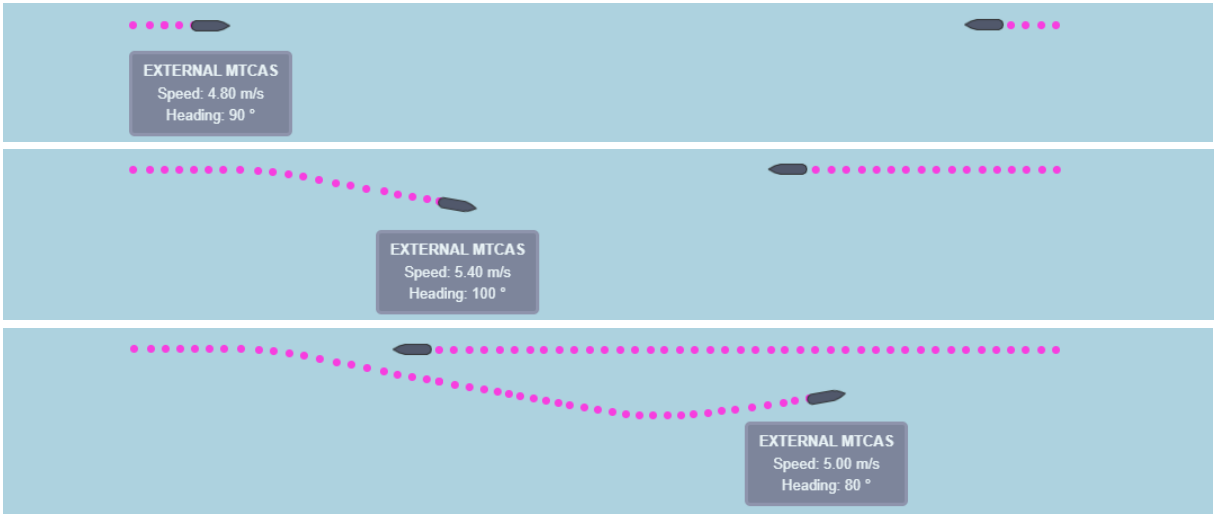


Abbildung 111: Screenshots der *WebView* während eines Simulationslaufs der beschriebenen Fallstudie. Das von links kommende Schiff repräsentiert die von der MTCAS-Implementierung gelieferten Daten. Das von rechts kommende Schiff wird durch DisCo-BaTS simuliert und seine Positionsdaten werden an die externe MTCAS-Implementierung kommuniziert. Oben: Zustand des Simulationslaufs zum Simulationszeitpunkt $t = 150$; Mitte: Zustand des Simulationslaufs zum Simulationszeitpunkt $t = 475$; Unten: Zustand des Simulationslaufs zum Simulationszeitpunkt $t = 1100$.

Abbildung 112: Screenshots der *WebView* während eines Simulationslaufs der beschriebenen Fallstudie zum Simulationszeitpunkt $t = 1025$. Das von links kommende Schiff repräsentiert die von der MTCAS-Implementierung gelieferten Daten. Das zugrundeliegende Szenario umfasst keine eigenständige Instanz eines Schiff-Elements. Da somit kein Bewegungsdaten erzeugt und an die externe MTCAS-Implementierung geliefert werden, leitet diese folgerichtig kein Ausweichmanöver ein.

Die XML-Repräsentation der erstellten Szenarioinstanz wurde anschließend als Eingabeparameter für den Start der Simulationsanwendung genutzt. Während des Simulationslaufs wurde die Entwicklung der durch den *GlobalObserver* entsprechend der Konfiguration als Teil der Szenarioinstanz ausgegebenen Größen des aktuellen Gesamtzustands mittels *WebView* (vgl. Abschnitt 15) in einem Webbrowser auf einem zusätzlichen Computersystem beobachtet. Die Entwicklung des Simulationszustands ist in Abbildung 111 anhand von drei Ausschnitten der Oberfläche der *WebView* dargestellt. Zu erkennen ist, dass die extern angebundene MTCAS-Implementierung wie erwartet auf das entgegenkommende Schiff reagiert, ein Ausweichmanöver einleitet, dieses durchführt und, sobald die Situation aufgelöst wurde, wieder zur ursprünglichen Route zurückkehrt. Durch die Durchführung einer Gegenprobe wurde anschließend untersucht, ob das von der MTCAS-Implementierung gezeigte Verhalten tatsächlich eine Reaktion auf die von *DisCo-BaTS* gelieferten Daten ist: Eine zweite Szenarioinstanz wurde erstellt, welche größtenteils deckungsgleich zur vorigen ist, aber kein eigens simuliertes Schiff (d. h., das *TargetShip* wurde aus der XML-Repräsentation entfernt) enthält. Bei simulativer Ausführung dieses zweiten Szenarios führt die MTCAS-Implementierung kein Ausweichmanöver durch (vgl. Abbildung 112). Die Kombination aus Probe und Gegenprobe hat somit bewiesen, dass die Abbildung von realen Testfällen durch Erstellung von Modellen und auf diesen basierenden Szenarioinstanzen sowie die Integration eines realen Testobjekts flexibel möglich sind.

16.3 UNTERSUCHUNG DES MEHRWERTS DES PROTOTYPS

Als letzter Untersuchungsbereich der Evaluation wird abschließend der Mehrwert als zusammenfassende, objektiv argumentative Einordnung der zusätzlichen Vorteile gegenüber einem üblichen Referenzvorgehen verdichtet dargestellt. Dieser Teil fällt bewusst knapper aus, da er sich wesentlich aus den Ergebnissen der beiden vorangehenden Teiluntersuchungen ableitet.

Die in den vorangegangenen Abschnitten exemplarisch untersuchten Anwendungsfälle sind selbstverständlich nicht auf dem Komplexitätsniveau eines produktiven Einsatzes, veranschaulichen jedoch die Eignung und das Potenzial des erarbeiteten Lösungskonzeptes. Es wurde u. a. gezeigt, dass es möglich ist, die meisten manuellen Schritte zu automatisieren, die für die Implementierung und Ausführung von HLA-konformen verteilten Simulationen erforderlich sind. Konkret sind mit dem prototypisch implementierten Ansatz im Rahmen der Erstellung von Modellen und Szenarioinstanzen sowie der simulativen Ausführung dieser keine Kenntnisse bezüglich des HLA-Standards selbst erforderlich: weder vom Modellentwickler, der die domänenspezifische Bibliothek implementiert, noch vom Tester, der diese zur Spezifikation von Szenarioinstanzen nutzt und dann simulativ ausführt. Die automatische Generierung der OMT-konformen FOM-Module ist dabei ein besonders zentraler Schritt.

Der IEEE-Standard 1730 *Recommended Practice for Distributed Simulation Engineering and Execution Process* (DSEEP) [IEEE, 1730:2022], der Nachfolger des inzwischen inaktiven IEEE-Standards *Recommended Practice for High-Level Architecture (HLA) Federation Development and Execution Process* (FEDEP) [IEEE, 1516.3:2003], beschreibt einen siebenstufigen Prozess zur Entwicklung und Ausführung verteilter Simulationsläufe. Dieser Prozess reicht von der Definition der übergeordneten Ziele, die die Simulationsumgebung erfüllen soll, bis zur Analyse der während der Simulationsläufe generierten Daten. Der hier vorgestellte und implementierte neuartige ganzheitliche Ansatz kann auf eine Mehrzahl von Schritten des DSEEP-Prozesses abgebildet werden. Während die meisten verwandten Ansätze und Frameworks für den Benutzer lediglich zwei bis drei Schritte zwischen den Schritten 2 und 4 vereinfachen oder vollständig ausblenden, lassen sich eine mögliche Verwendung des präsentierten ganzheitlichen Ansatzes und dessen prototypischer Umsetzung in den Schritten 2 bis 6 erkennen. Dies bietet das Potenzial, die Konsistenz des Gesamtprozesses zu erhöhen und gleichzeitig die Komplexität zu verringern. Abbildung 113 stellt den DSEEP-Prozess, wie er im Standard beschrieben ist, und einen durch die Integration des hier vorgestellten Ansatzes erweiterten Prozess gegenüber. Es ist zu erkennen, dass die ursprünglichen Schritte 3–6 ersetzt wurden: Aus Sicht eines Benutzers entfällt Schritt 5 vollständig und die Schritte 3, 4 und 6 werden vereinfacht.

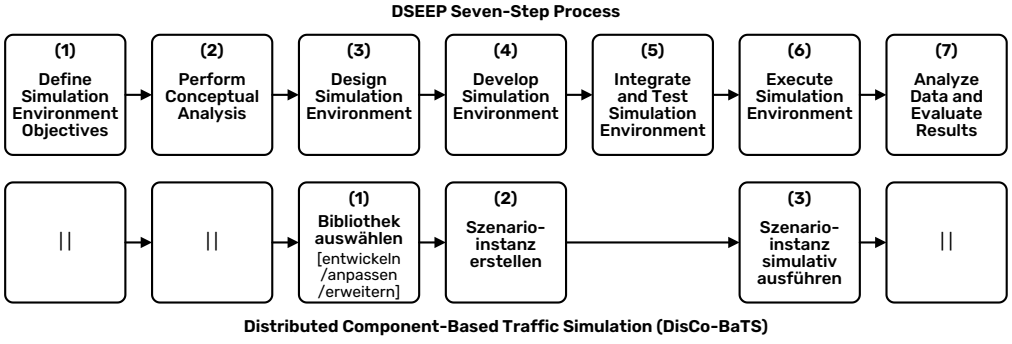


Abbildung 113: Integration des erarbeiteten Lösungsansatzes in den *Distributed Simulation Engineering and Execution Process* (DSEEP) zur Umsetzung und Nutzung einer *High Level Architecture* (HLA) Föderation.

Das erarbeitete Konzept sieht vor, dass der Entwickler der auf den Modellzweck ausgerichteten Modellbibliothek eine andere Person bzw. Personengruppe ist (Modellentwickler) als diejenige, die für die Erstellung von Szenarien verantwortlich ist (Tester). Durch diese Aufteilung des Wissens und der Arbeitslast wird auch die Verantwortung für das Sicherstellen der Korrektheit der Modelle auf Erstere übertragen. Dies gilt zumindest für die Funktionalität der möglichen Modell-Elemente, die Teil der Bibliothek und der darunterliegenden Modell-Ebenen sind. Ob ein bestimmter Simulationslauf wie geplant abläuft, ist weiterhin Aufgabe des Testers, was jedoch nicht mit einem Test der Modelle und der Simulationsanwendung auf technische Fehlerfreiheit gleichgesetzt werden kann. Der Startprozess, ursprünglich Schritt 6, ist so weit automatisiert, dass das Simulationssystem direkt mit einer Szenarioinstanz als einziger Eingabe gestartet werden kann, was eine der Hauptanforderungen war, um einen szenariobasierten Ansatz im Kontext simulativer V&V angemessen zu unterstützen.

16.4 ZUSAMMENFASSENDE UNTERSUCHUNG DER ZIELERREICHUNG

Die Ergebnisse der vorangegangenen Untersuchungen zusammenfassend und somit die Bewertung abschließend werden nachfolgend die in Abschnitt 12 definierten gestaltungsorientierten Ziele in Hinblick darauf betrachtet, ob diese von den erstellten Konzepten und dem umgesetzten Prototyp erreicht werden.

Teilziel 1: Entwicklung einer einheitlichen Basis zur Erstellung flexibel wiederverwendbarer und erweiterbarer Modelle maritimer Verkehrsszenarien sowie zur Spezifikation von Instanzen dieser Modelle.

Die Erfüllung aller diesem Teilziel zuzuordnenden gestaltungsorientierten Anforderungen wurde in den Abschnitten 16.1.1 und 16.1.2 hinreichend argumentativ mit Hilfe von Rückbezügen zu den entsprechenden Lösungskonzepten und den konkreten Ausgestaltungen dieser aufgezeigt. Die in Abschnitt 16.2.1.1 aufgeführten Untersuchungen konnten darüber hinaus einen hohen Grad an Wiederverwendbarkeit sowie die flexible Erweiterbarkeit belegen und somit die Eignung für nötige Anpassungen an den jeweiligen Modellzweck nachweisen.

Teilziel 2: Entwicklung einer Anwendung zur direkten Durchführung simulativer Testläufe basierend auf Instanzen heterogener Szenariomodelle.

Die diesem Teilziel zugeordneten gestaltungsorientierten Anforderungen wurden in Abschnitt 16.1.3 erfolgreich auf Erfüllung überprüft. Die der Erreichung dieses Teilziels zuträglichen erarbeiteten Lösungskonzepte konnten in der Gesamtheit der Untersuchung des erstellten Prototyps in den Abschnitten 16.2.1 und 16.2.2 ihre Eignung sowie die Erreichung des genannten Teilziels unter Beweis stellen.

Teilziel 3: Entwicklung eines Konzepts zur Kapselung der Komplexitäten zugrunde liegender abstrakter Strukturen der Modelle und technischer Abläufe der Simulationsanwendung.

Als inhaltlich über alle Teilaspekte der entwickelten Lösungsbestandteile übergreifendes Ziel wurden die dem Teilziel 3 zugeordneten Anforderungen in den Abschnitten 16.1.1, 16.1.2 und 16.1.3 verteilt als erfüllt bestätigt. In Abschnitt 16.2.1.2 konnte die notwendige Kapselung von Komplexität und die damit verbundene klare Abgrenzung von Rollen im Prozess szenariobasierten simulativen Testens am Beispiel der zuvor dargestellten praktischen Untersuchungen der Kernkonzepte in Abschnitt 16.2.1 nachgewiesen werden. Auch die Fallstudie in Abschnitt 16.2.2 zeigt eindrücklich, dass zur simulativen

Durchführung konkreter Testfälle und den damit einhergehenden vorgelagerten Tätigkeiten bezüglich der Modelle und Szenarioinstanzen keine Kenntnisse zu den spezifisch eingesetzten Technologien den definierten Rollen entsprechend nötig sind. Die Untersuchung des Mehrwerts des Prototyps 16.3 legt ebenfalls nahe, dass die Gesamtkomplexität des Entwicklungsprozesses einer verteilten Simulationsanwendung durch den Einsatz des Prototyps abnimmt.

Teilziel 4: Entwicklung eines Konzepts zur Integration heterogener zu testender Komponenten in Instanzen des Szenariomodells.

Die Einbindung von Testobjekten unterschiedlicher Ausprägung in konkrete Szenarioinstanzen wird durch die Ausgestaltung des zugrundeliegenden Metamodells ermöglicht (vgl. Abschnitt 14.3.2.5). Der Nachweis der Erfüllung der entsprechenden gestaltungsorientierten Anforderung wurde in Abschnitt 16.1.2 erbracht. Die Integration einer umgesetzten Testanwendung in verschiedenen Konfigurationsvarianten in Abschnitt 16.2.1.3 sowie die Integration eines realen Assistenzsystems im Rahmen der Fallstudie in Abschnitt 16.2.2 konnten anschließend auch die praktische Eignung und somit die Erreichung des Ziels nachweisen.

Teilziel 5: Entwicklung eines Konzepts zur aktiven Einbindung heterogener externer Komponenten in den durch eine Szenarioinstanz repräsentierten Simulationslauf.

Anknüpfend an die Ergebnisse der Überprüfung des Teilziels 4 auf Erreichung dessen kann auch Teilziel 5 als erreicht bestätigt werden. Der Nachweis über die Erfüllung der in Bezug zu diesem stehenden gestaltungsorientierten Anforderungen wurde in 16.1.3 erbracht. Die anschließenden Untersuchungen der individuellen Teilkonzepte sowie die gesamtheitliche Untersuchung des Prototyps im Rahmen der konkreten Fallstudie auf Eignung haben die Fähigkeit des Prototyps, vielfältigen Anforderungen unterschiedlicher externer Testobjekte gerecht zu werden und eine vollständige Einbindung in den Simulationslauf im Sinne von syntaktischer und semantischer Interoperabilität durch flexible bidirektionale Kommunikation zu ermöglichen, eindrücklich veranschaulicht.

Auf Grundlage der Gesamtheit der Nachweise zur Anforderungserfüllung, der Eignung des Prototyps, dem Mehrwert des Prototyps sowie der Erreichung der individuellen Teilziele kann ebenfalls das übergeordnet formulierte Ziel des konstruktiven zweiten Teils dieser Arbeit als erreicht bestätigt werden: „*Entwicklung eines softwaretechnischen Konzepts, welches eine einheitliche szenariobasierte Durchführung submikroskopischer Simulationsläufe zur Verifizierung und Validierung softwarebasierter maritimer Fahrsysteme entwicklungsbegleitend ermöglicht.*“

Logisch schlussfolgernd kann auf Grundlage des Erreichens aller aufgestellten Ziele an dieser Stelle nun auch die am Ende von Abschnitt 12 aufgestellte Hypothese als verifiziert angesehen werden: Der Kern der erarbeiteten Lösungskonzepte, die Auflösung der orthogonalen Beziehung zwischen Szenario- und Simulationsmodell, hat die Grundlage geschaffen, um in Kombination mit einer stringenten Kapselung etwaiger Komplexitäten die Erschaffung eines Softwarewerkzeugs in Form des umgesetzten Prototyps zu ermöglichen, das für die systematische szenariobasierte Durchführung entwicklungsbegleitender simulativer Tests im Kontext der Verifizierung und Validierung geeignet ist. Die in Abschnitt 14 präsentierten Lösungskonzepte und die entsprechenden inhaltlichen Ausgestaltungen stellen somit eine Antwort auf die eingangs in Abschnitt 3 aufgestellte Forschungsfrage dar, deren Beantwortung sich diese Arbeit gewidmet hat.

IV Zusammenfassung

Die flächendeckende Einführung hochgradig automatisierter Wasserfahrzeuge, insbesondere derer, die durch den Transport von Gütern und Waren das Rückgrat der modernen globalen Wirtschaft bilden, bietet viele Potenziale. Auf dem Weg zur öffentlichen Zulassung entsprechender Systeme, Teilsysteme und Komponenten sind aktuell allerdings noch einige Hürden zu überwinden. Eine davon ist der hinreichend verlässliche Nachweis der sicheren und korrekten Ausführung eben jener automatisierter Funktionalitäten im Rahmen einer systematischen Verifizierung und Validierung (V&V). Im Rahmen dieser Arbeit wurde auf Grundlage ingenieurwissenschaftlicher Forschungsmethodik ein softwaretechnisches Artefakt geschaffen, das Lösungen für zentrale Herausforderungen der Integration szenariobasierter, simulativ durchgeführter Tests in einen V&V-begleiteten Entwicklungsprozess bietet.

Zur Strukturierung des Forschungsprozesses wurde ein zielgerichtetes, gestuftes Vorgehen gewählt, das den Übergang von einer anfänglich offenen Problemstellung hin zu einer methodisch fundierten und konstruktiven Lösungsentwicklung nachvollziehbar macht (vgl. Abbildung 3 und Abbildung 4). Ausgangspunkt war ein zunächst unscharf umrissenes Themenfeld (vgl. Teil I), dessen genaue Problemstrukturen nicht unmittelbar bestimmbar waren. Um diese Unschärfe zu überwinden, wurde zunächst eine explorative Vorstudie (vgl. Teil II) durchgeführt. Ziel dieser Vorstudie war es, einen systematischen Überblick zu gewinnen, bestehende Ansätze und Herausforderungen zu identifizieren und eine Problemfokussierung abzuleiten.

Auf Basis dieser Erkenntnisse konnten im konstruktiven Teil dieser Arbeit (vgl. Teil III) zunächst gestaltungsorientierte Zielsetzungen (vgl. Abschnitt 12) formuliert werden. Diese Zielsetzungen definierten den spezifischen Forschungsbeitrag im Sinne einer konstruktiven Arbeit und bildeten die inhaltliche Grundlage für die weitere Entwicklung. Ergänzend hierzu wurden aufbauend auf den übergeordneten allgemeinen Anforderungen an den generellen Einsatz szenariobasierter maritimer Verkehrssimulation (vgl. Abschnitt 9) anhand der gewonnenen Erkenntnisse spezifische lösungsbezogene Anforderungen abgeleitet (vgl. Abschnitt 13). Diese legten einerseits die Rahmenbedingungen der Konstruktion fest und stellten andererseits Maßstäbe zur Bewertung der angestrebten Lösung bereit. Mit diesen gestaltungsorientierten Zielen und Anforderungen als Fundament erfolgte anschließend der konstruktive Lösungsentwurf (vgl. Abschnitt 14). In diesem Schritt wurde die Forschungsidee in ein konkretes Konzept überführt, wobei die getroffenen Gestaltungsentscheidungen nachvollziehbar hergeleitet und anschließend gegenüber alternativen Ansätzen abgegrenzt wurden. Abschließend wurde die entworfene Lösung einer Bewertung hinsichtlich ihrer Praxistauglichkeit (vgl. Abschnitt 16) unterzogen. Hierbei wurde anhand einer prototypischen Implementierung der Konzepte überprüft, ob die zuvor definierten Anforderungen an die konstruktive Lösung erfüllt wurden, ob und wie die gestaltungsorientierten Zielsetzungen durch die konstruierte Lösung erreicht wurden und in welchem Maße sich die erarbeitete Lösung für den Einsatz im Bereich der Eingangsproblemstellung eignet.

Das dargestellte Vorgehen verdeutlicht den Charakter der Arbeit als ingenieurwissenschaftlicher Forschungsprozess: von einer offenen Exploration über die systematische Konkretisierung bis hin zur zielgerichteten Konstruktion einer Lösung und praxisnahen Bewertung dieser. So wurde ein nachvollziehbarer Forschungsbogen gespannt, der sowohl wissenschaftliche Stringenz als auch praktische Relevanz sicherstellt.

17 EINORDNUNG DES FORSCHUNGSBEITRAGS

Den wesentlichen Forschungsbeitrag stellt die ganzheitliche Betrachtung des szenariobasierten Einsatzes von Computersimulationen zur Durchführung virtueller Testläufe dar. Trotz des hohen wissenschaftlichen Interesses und der wirtschaftlichen Relevanz ist die gemeinsame Betrachtung der beiden Themenfelder der *Abbildung von Testfällen auf Basis von Verkehrsszenarien* und die *Durchführung von Verkehrssimulationen* kaum anzutreffen. Während der Anfertigung dieser Arbeit konnten keine Arbeiten ausfindig gemacht werden, die beide Themenfelder gleichgewichtet, kombiniert und harmonisierend betrachtet haben. Dieser Tatsache ist es geschuldet, dass die Überführung von identifizierten Szenarien in einen Simulationslauf bisher stets ein Kompromiss war, welcher u. a. die Notwendigkeit manueller Modelltransformationen oder hohe Abhängigkeiten mit sich brachte. Durch die ganzheitliche Betrachtung beider Themenfelder mit explizitem Fokus auf die Überführung einer Szenariobeschreibung in einen Simulationslauf und einer Zusammenführung und Harmonisierung der Anforderungen an die drei thematischen Bereiche *Szenario*, *Simulation* sowie der erwähnten *Überführung* konnte ein technisches Lösungskonzept gestaltet werden, welches auf die Unterstützung des gesamten Prozesses ausgerichtet ist. Nachfolgend werden die relevantesten Aspekte der erarbeiteten Konzepte bezüglich ihres Beitrags genauer betrachtet.

So erfuhr die wichtige flexible Integration von zu testenden Systemen, Teilsystemen und Komponenten bisher keine Aufmerksamkeit in der Art, wie sie in dieser Arbeit als notwendig identifiziert wurde. Testmethoden wie *Model-in-the-Loop (MiL)*, *Software-in-the-Loop (SiL)*, *Hardware-in-the-Loop (HiL)* und *Vehicle-in-the-Loop (ViL)* (vgl. Abschnitt 7.3.4) adressieren Aspekte des Themengebiets zwar ausgiebig, aber Ansätze für die flexible Ermöglichung der Integration unterschiedlicher Ausprägungen von zu testenden Systemen in Simulationsläufe im Sinne syntaktischer und semantischer Interoperabilität sind selten anzutreffen. Die vorliegende Arbeit liefert eine Möglichkeit, sowohl lokale als auch entfernte Testobjekte semantisch flexibel an Simulationsläufen teilnehmen zu lassen.

Ebenso bietet das erarbeitete Metamodell das Potenzial, als Basis für einheitliche, austauschbare und vergleichbare Szenariospezifikationen zu dienen. Eine solche existiert im Bereich der maritimen Simulationen bisher nicht. Im Automobilbereich erfüllt der weitverbreitete offene Standard OpenSCENARIO diese Aufgabe. Aufgrund der Tatsache, dass dieser ohne direkten Bezug zu einer simulativen Ausführung entwickelt wurde, existieren zwar eine Vielzahl an Simulationen, die die entsprechenden Szenariospezifikationen interpretieren können, aber die als problematisch identifizierte bidirektionale orthogonale Abhängigkeit zwischen dem Szenario- und dem Simulationsmodell erschwert die flexible Adaption an konkrete Testfälle und Modellzwecke.

Zusammengefasst liefert diese Arbeit einen wertvollen Beitrag für die effiziente simulative Durchführung von Tests im Rahmen der Verifizierung und Validierung softwarebasierter Systeme und Systemkomponenten, indem die sechs in Abschnitt 11 identifizierten Handlungsbedarfe explizit adressiert

wurden und die vorgestellten Konzepte für jede dieser bisher existierenden thematischen Lücken einen möglichen Lösungsansatz bieten.

- Der gesamte V&V-integrierte Entwicklungsprozess wird durch die gebotene Flexibilität in jeder Phase unterstützt.
- Das erarbeitete Metamodell bietet eine einheitliche Grundlage für die technische Modellierung von Seeverkehrsszenarien und bietet somit das Potenzial, die Zusammenarbeit und Kommunikation zwischen Projekt- und Entwicklungspartnern zu fördern.
- Die Simulationsanwendung auf Basis des Konzepts des transitiven Simulationsmodells ermöglicht die simulative Ausführung von Szenarien unterschiedlicher Komplexitäts- und Abstraktionsgrade. Dabei kommt es weder zu Informationsverlust noch sind manuelle Transformationen nötig.
- Der entstandene Prototyp als Umsetzung der erarbeiteten Lösungskonzepte bietet das Potenzial, ein effizientes entwicklungsbegleitendes Werkzeug zu sein, da Komplexitäten, mit denen ein potenzieller Anwender in Kontakt kommt, effektiv reduziert wurden. Die Voraussetzung für die Übertragbarkeit dieses Aspekts auf die Realität ist die klare Trennung der Verantwortlichkeiten entsprechend der definierten Benutzerrollen.
- Das Konzept und der umgesetzte Prototyp bieten durch die gezielte Abbildung komponentenbasierter Wirkstrukturen und die flexible Möglichkeit der Integration und Konfiguration des Testobjekts in diese die Möglichkeit, das Testobjekt als Bestandteil des Szenarios zu integrieren.
- Der konfigurierbare, dreistufige Kommunikationsprozess ermöglicht flexible bidirektionale Kommunikation mit externen Testobjekten im Sinne syntaktischer und semantischer Interoperabilität.

Die durch die Gesamtheit der erarbeiteten Konzepte gebotene Erweiterbarkeit und Wiederverwendbarkeit bietet trotz des einheitlichen Metamodells eine Grundlage für zukünftige einheitliche Umsetzungen virtueller Testfelder vielfältiger Ausprägungen. Denn für derartige Testfelder unabdingbare Bestandteile wie Sensoren, Sensordatenverarbeitung und Wahrnehmung können in die Modell- und Ausführungsstrukturen integriert werden, wie anhand einfacher Beispiele im Rahmen der Untersuchung der Eignung gezeigt wurde. Standardisierte Daten- und Kommunikationsformate lassen sich ebenso unkompliziert integrieren, wie ebenfalls im Rahmen der Untersuchung der Eignung und der Fallstudie demonstriert wurde. Solche Testfelder sind unabdingbar für die künftige öffentliche breite Zulassung von hochautomatisierten Wasserfahrzeugen [Wright, 2020]. Wahrscheinliche positive Effekte eines solchen Einsatzes sind eine Reduzierung von Entwicklungszeiten softwarebasierter Komponenten automatisierter Fahrzeuge, eine Erhöhung der Sicherheit durch eine steigende simulative Testabdeckung sowie ein erhöhtes Maß an Kooperation von Entwicklungspartnern und zuliefernden Fremdunternehmen im Kontext der Entwicklung moderner Schiffe als System-of-Systems.

18 REFLEXION DER ENTWICKELTEN LÖSUNG

Obwohl die vorgestellten Konzepte und die prototypische Umsetzung das Potenzial haben, den Aufwand für die technische Spezifikation und simulative Ausführung von vielen Arten von Verkehrsszenarien deutlich zu reduzieren, weisen diese sowohl konzeptionelle als auch technische Grenzen auf, die die Anwendbarkeit teilweise einschränken.

Durch die vollständige Ausblendung der HLA-Schnittstellendetails für Entwickler und Anwender können die Vorteile der HLA-basierten verteilten Co-Simulation genutzt werden, ohne dass man sich

mit den komplexen HLA-Prozessen und -Strukturen vertraut machen muss. Viele Mechanismen des gezeigten Konzepts und der Implementierung wurden speziell auf submikroskopische Verkehrssimulationen ausgerichtet entwickelt. Die entworfene vielschichtige Modellhierarchie erlaubt zwar grundsätzlich, den gezeigten Ansatz ebenfalls auf andere Betrachtungsebenen und Domänen zu übertragen. Die Eignung für andere Verkehrsarten oder gar gänzlich domänenfremde Einsatzbereiche wurde allerdings nicht erprobt und nicht in allen Teilkonzepten explizit angedacht.

Die erarbeiteten Lösungskonzepte sind aus nachvollziehbar dargestellten Gründen vollständig auf zeitdiskrete Modelle und Simulationsabläufe ausgerichtet. Diese Tatsache erschwert die Integration kontinuierlicher Modelle oder macht sie ohne die Anpassung grundlegender Mechanismen im aktuellen Zustand sogar unmöglich. Da softwarebasierte Abläufe aufgrund der diskreten Natur üblicher Prozessoren selbst nicht vollständig kontinuierlich sein können, stellt dies für die Modellierung der Fahrzeuge und ihrer internen Wirkstrukturen in der Regel kein Problem dar. Die Integration von Modellen der Fahrzeugdynamik oder von Wetterphänomenen wird somit aber potenziell eingeschränkt.

Im Rahmen der Konzepterstellung wurden einige vereinfachende Annahmen getroffen oder komplexe Randfälle bewusst ausgeblendet, um die Umsetzung flexibler grundlegender Strukturen und Abläufe zu ermöglichen. So ermöglicht etwa das Konzept des *Dictionary* auf eine generische Art und Weise die Möglichkeit zur konfigurationsbasierten, semantisch interoperablen, bidirektionalen Kommunikation mit externen Testobjekten. Allerdings beschränken sich die Möglichkeiten hier aktuell auf Umrechnungen zwischen verschiedenen einfachen Einheiten und Einheitenpräfixen. Ob und wie eine Übersetzung von einer Positionsangabe mit Bezug zu einem planaren kartesischen Koordinatensystem in eine dem geozentrischen Bezugssystem World Geodetic System 1984 (WGS84) entsprechende in diesen Mechanismus integriert werden könnte, müsste zunächst klärend untersucht werden.

19 ANKNÜPFENDER HANDLUNGS- UND FORSCHUNGSBEDARF

> *„There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity."*
>
> [Brooks, 1987]

Aufgrund ihrer Ausrichtung erstreckt sich diese Arbeit auf ein außerordentlich weites Themenfeld und stellt keineswegs eine geschlossene oder gar vollendete Lösung für alle Fragen im Zusammenhang mit zukünftigen simulationsbasierten Testprozessen moderner Assistenz- und Fahrsysteme dar. Im Gegenteil: Vor allem aufgrund des prototypischen Zustandes der entwickelten Lösung im Sinne eines Proof-of-Concept existiert eine Vielzahl von Möglichkeiten zur Verbesserung, Optimierung und Erweiterung in verschiedene Richtungen, von denen einige im Folgenden skizziert werden. Diese können als mögliche Ausgangspunkte für anknüpfende Forschungsvorhaben dienen.

Technische Aspekte

Die Performance und der Ressourcenbedarf hatten während der Erarbeitung der Konzepte und der experimentellen Umsetzung lediglich einen geringen Stellenwert. Davon, wie performant eine Simulationsanwendung ist, hängt schlussendlich aber der relevante Faktor der maximalen Beschleunigung im Verhältnis zur Echtzeit ab. Wie performant die Ausführung umfangreicher Simulationsläufe mit vielen Remote-Federates, einer großen Anzahl unterschiedlicher Remote-Hosts und/oder einer Vielzahl ex-

tern angebundener Simulationsteilnehmer ist, muss daher systematisch untersucht werden. Ebenso sollte die Auswirkung unterschiedlicher Netzwerkausprägungen, -geschwindigkeiten und -laufzeiten untersucht werden, damit ggf. passgenaue Optimierungsmaßnahmen ergriffen werden können.

Zwei mögliche Ansatzpunkte zur Optimierung der Ausführungsperformanz mit hohem Potenzial wurden während der Erarbeitung der Konzepte und der experimentellen Umsetzung identifiziert:

- Die Integration von *Region of Interest* (ROI) durch Nutzung des vom HLA-Standard vorgesehenen *Data Distribution Management* (DDM)-Service [IEEE, 1516:2010; Petty, 2002; Morse & Steinman, 1997] zur räumlichen Definition von Bereichen innerhalb der simulierten Umwelt, auf welche die *Subscriptions* von Objekten und Attributen beschränkt werden. Auf die Art könnten u. a. beschränkte Sichtweiten umgesetzt und die Menge der innerhalb der verteilten Simulationsanwendung übertragenen Daten könnte deutlich reduziert werden. Andere mögliche HLA-spezifische Ansätze zur Performanceoptimierung wurden u. a. in [Santoro & Fujimoto, 2008], [Thomas Roth et al., 2018] und [Tan & Xu, 2001; Tan et al., 2000] vorgestellt.
- Die Gesamtperformanz eines Simulationslaufs kann möglicherweise stark davon profitieren, wenn die Möglichkeit bestünde, dass die Gesamtsimulation langsamer in der Simulationszeit voranschreitet als die Verhaltensimplementierung innerhalb der Federates. Ein solcher Ablauf könnte die Netzwerklast verringern bei gleichzeitiger Erhaltung der Genauigkeit von Verhaltensfortrechnungen im Sinne von Federate-internen *Direct Feedthrough*-Schleifen.

Automatisierte Fahrfunktionen, insbesondere dann, wenn diese mit integrierten Steuergeräten im Verbund in einem Systemverbund interagieren, müssen häufig quantitative Echtzeitanforderungen erfüllen [Brinkmann, 2018]. Die prototypische Umsetzung der erarbeiteten Konzepte in Form des Frameworks *DisCo-BaTS* wurde unter Verwendung der Programmiersprache *Java* umgesetzt. Diese erfüllt in der Standardausführung nicht die Anforderungen an eine mögliche Echtzeitfähigkeit. Ferner erfolgt die Kommunikation mit externen Testobjekten im Rahmen des erarbeiteten Konzepts häufig über das Internet als Kommunikationsmedium und Kommunikationsprotokolle wie TCP. Beide Technologien sind in der Regel ebenfalls nicht echtzeitfähig. Eine Alternative zur Echtzeitfähigkeit kann in vollständig simulationsbasierten Umgebungen sein, einen Rollbackmechanismus zu etablieren, der bei auftretenden Problemen einen vorigen Zustand wiederherstellt und von diesem aus erneut mit der Fortschreitung beginnt. Aufgrund der Integration von Testobjekten, die diese Funktion in den allermeisten Fällen nicht unterstützen und nicht direkt kontrolliert werden können, stellt ein solcher Mechanismus hier keine mögliche Lösung dar. Stattdessen müssten Möglichkeiten identifiziert, untersucht und ggf. integriert werden, die eine echtzeitfähige simulationsinterne Verarbeitung und eine echtzeitfähige Kommunikation zwischen den Simulationsteilnehmern ermöglichen können. Mögliche Ansatzpunkte könnten die *Real-Time Specification for Java* (RTSJ) oder der Einsatz eines *Software-Defined WAN (SD-WAN)* sein.

Der aktuelle Stand der prototypischen Umsetzung setzt als Format zur Serialisierung der Szenarioinstanzen XML ein. XML als Format zum Übertragen und Persistieren von Daten ist zwar weit verbreitet, aber es existiert eine Vielzahl konkurrierender textbasierter Formate. Als Beispiel sei an dieser Stelle YAML genannt, das einige potenzielle Vorteile gegenüber XML bietet, wie eine deutlich bessere Menschenlesbarkeit und eine flexiblere Strukturierung von Daten, insbesondere von komplexen Datentypen, bei gleichzeitigem Erhalt vieler der Vorteile von XML, wie die gute Maschinenlesbarkeit und

der hierarchische Aufbau. Alternative textbasierte Formate sollten entsprechend auf ihre Eignung untersucht werden und XML ggf. ablösen.

Inhaltliche Aspekte

Naheliegende Erweiterungsmöglichkeiten sind in der Erweiterung der Einsatzmöglichkeiten auf vor- und nachgelagerte Teilprozesse eines typischen szenariobasierten Entwicklungsprozesses (vgl. Abbildung 2) zu finden. Innerhalb eines solchen szenariobasierten Entwicklungsprozesses automatisierter Fahrfunktionen werden im Rahmen der inhaltlichen Spezifikation der Szenarien üblicherweise zunächst logische Szenarien definiert (vgl. Tabelle 6). Um die Unterstützung des Gesamtprozesses weiter auszuweiten, wäre es wünschenswert, wenn die technische Spezifikation in Form einer Szenarioinstanz, wie sie in dieser Arbeit vorgestellt wurde, nicht nur konkrete, sondern auch logische Szenarien repräsentieren könnte. Das heißt konkret, dass das vorgestellte Konzept und die prototypische Implementierung dahingehend erweitert werden sollten, dass den Eigenschaftsfeldern der Szenarioelemente zusätzlich zu konkreten Werten auch Wertebereiche zugewiesen werden können. Grundlegende dafür nötige Strukturen sind schon im Programmcode der prototypischen Umsetzung enthalten: Ein *Property*-Objekt kann bereits Listen von Objektinstanzen als *value* entgegennehmen. Die Entwicklung und Umsetzung eines entsprechenden Ablaufs, der auf Basis innerhalb eines Szenarios vorhandener Wertebereiche eine Menge möglicher Permutationen der jeweiligen Belegungen erzeugt und diese automatisiert aufeinanderfolgend ausführt, stellen hingegen einen möglichen Anknüpfungspunkt zukünftiger Bemühungen dar. Auch die Evaluation der Systemperformanz auf Basis der Zustandstrajektorie eines Simulationslaufs kann möglicherweise teilweise in den Modellierungs- und Ausführungsprozess einer Szenarioinstanz integriert werden. Zu untersuchen ist dafür, wie die Definition von erwartetem Verhalten in die durch das Metamodell gegebenen Strukturen integriert werden kann und wie diese mit dem Zustand während des Simulationslaufs abgeglichen werden kann. Das Ziel sollte dabei sein, dass Abweichungen von dem definierten erwarteten Verhalten im besten Fall direkt zur Laufzeit erkannt werden und der Test in Form eines Simulationslaufs als fehlgeschlagen eingestuft und abgebrochen wird. In [Klikovits et al., 2024] und [King et al., 2019] werden Ansätze vorgestellt, welche als Ausgangspunkte für eine entsprechende Untersuchung dienen könnten.

Die Ergebnisse dieser Arbeit ermöglichen die semantisch und syntaktisch interoperable Einbindung vielfältiger externer Testobjekte. Diese externen Simulationsteilnehmer müssen aktuell noch händisch und ggf. zu einem bestimmten Zeitpunkt gestartet werden, da bisher keine Möglichkeit umgesetzt wurde, den Startprozess des Simulationslaufs mit dem der externen Teilnehmer zu synchronisieren. Mögliche Ansatzpunkte bieten hier u. a. die Arbeit von [Gütlein, 2023] und das relativ junge *Distributed Co-Simulation Protocol (DCP)*, welches in Abschnitt 14.3.1 bereits kurz thematisiert wurde.

Trotz des Einsatzes eines systematischen Vorgehens wie dem szenariobasierten Testen wird es möglicherweise nicht vollständig vermeidbar sein, zusätzlich praxisnahe Abläufe zu testen. Damit sind an dieser Stelle Situationen und Abläufe gemeint, an denen eine große Anzahl Verkehrsteilnehmer teilnimmt, welche sich u. U. nicht vollständig deterministisch verhält und deren Existenz einer zufällig wirkenden Verteilung folgt. Eine entsprechende Szenarioinstanz zu spezifizieren, ist aktuell mit hohem Aufwand verbunden, da für Variation der Verkehrsdichte etliche Verkehrsteilnehmer händisch spezifiziert werden müssten. Die Integration eines Ansatzes zur parametrierbaren, automatisierten, gewissermaßen zufällig wirkenden Generierung von Verkehr sollte daher angestrebt werden. Ein möglicher

Ansatz könnte die Verteilung der Ankunftszeiten mittels eines *Poisson Arrival Process* (PAP) [Kim & Whitt, 2014; Faÿ et al., 2006]. Eine derartige Möglichkeit zur parametrierbaren automatisierten Erzeugung von Verkehr würde zudem die Menge der Einsatzmöglichkeiten auf Bereiche wie das Verkehrsmanagement erweitern.

Für den zuvor genannten Anwendungsfall könnten auch externe Anwendungen, welche bereits in der Lage sind, Verkehr oder Infrastruktur mittels Netzwerkkommunikation zur Verfügung zu stellen, angebunden und die gelieferten Inhalte anreichernd in den Simulationslauf integriert werden (vgl. Abbildung 114). Aber auch in diesem Fall ist die Menge der manuellen Arbeit aktuell hoch, da für jedes zugespielte Objekt ein Modellelement in die Szenarioinstanz integriert und mit einem entsprechend konfigurierten *ProxyBehaviour* versehen werden muss. Ein denkbarer Lösungsansatz könnte die Erweiterung des Metamodells um ein entsprechendes Konfigurationselement und des Modellüberführungsmechanismus um einen weiteren Federate-Typ sein. Die Integration in eine Szenarioinstanz könnte dann so ausgestaltet werden, dass, ähnlich der Konfiguration eines *Dictionary* zur Ermöglichung flexibler bidirektionaler Kommunikation, den vom externen Zuspieler erwarteten Objekttypen jeweils eine im Modell vorhandene Elementklasse zugeordnet wird. Bei Empfang eines Objektes von der externen Datenquelle könnte dann eine entsprechende Instanz erzeugt werden und durch den *Model-Aware RTI Exchange* Mechanismus den restlichen Simulationsteilnehmern bekannt gemacht werden. Die manuell nötige Arbeit würde so darauf reduziert werden, dass das Modell um ggf. nötige Repräsentationen der Arten von zu erwartenden Objekten erweitert und eine pro erwartetem Typ einmalige Zuordnung dieser Objekte und ihrer Attribute auf die Modellrepräsentationen in die Szenarioinstanz integriert wird. Eine derartige Erweiterung würde auch die Anbindung von Softwareanwendungen wie dem *Nautilus Chartserver*[106] ermöglichen, um große Mengen infrastruktureller Daten innerhalb des Simulationslaufs nutzbar zu machen (vgl. Abbildung 115) statt eigene umfangreiche Infrastrukturbeschreibungen in eine Szenarioinstanz integrieren zu müssen.

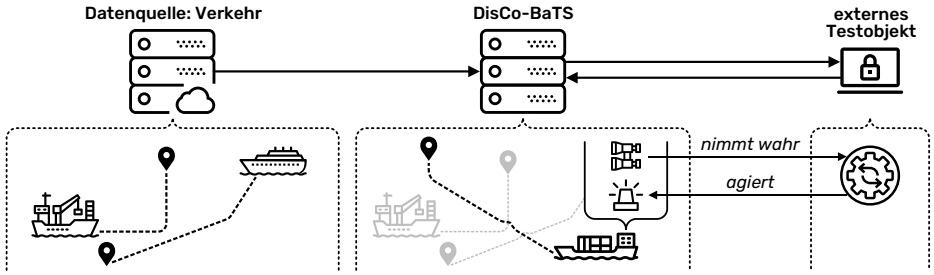


Abbildung 114: Erprobung eines Testobjekts bei gleichzeitiger Einspielung externer Verkehrsdaten

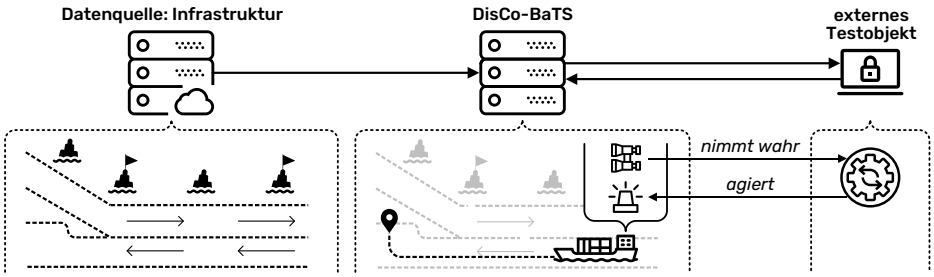


Abbildung 115: Mögliche Einspielung externer Infrastrukturdaten

[106] SevenCs GmbH, Nautilus ChartServer: https://www.sevencs.com/chartserver/nautilus-chartserver/ [letzter Zugriff: 17.02.26]

LITERATUR

Abar, S., Theodoropoulos, G. K., Lemarinier, P. & O'Hare, G. M. (2017): Agent Based Modelling and Simulation tools: A review of the state-of-art software. *Computer Science Review*, 24, S. 13–33. DOI: 10.1016/j.cosrev.2017.03.001.

Acciaro, M., Ferrari, C., Lam, J. S., Macario, R., Roumboutsos, A., Sys, C., Tei, A. & Vanelslander, T. (2018): Are the innovation processes in seaport terminal operations successful? *Maritime Policy & Management*, 45 (6), S. 787–802. DOI: 10.1080/03088839.2018.1466062.

Adam, C. & Arduin, H. (2024): Agent-based epidemics simulation to compare and explain screening and vaccination prioritization strategies. *SIMULATION*, 100 (4), S. 335–355. DOI: 10.1177/00375497231178303.

Adnan, M., Pereira, F., Lima Azevedo, C., Basak, K., Lovric, M., Raveau, S., Zhu, Y., Ferreira, J., Zegras, C. & Ben-Akiva, M. (2016): SimMobility: A Multi-Scale Integrated Agent-based Simulation Platform. In *95th TRB Annual Meeting:* National Academies of Sciences, Engineering and Medicine.

Akkermann, A. & Hjollo, B. A. (2019): Scenario-Based V&V in a Maritime Co-Simulation Framework. In *2019 Spring Simulation Conference (SpringSim 2019):* Institute of Electrical and Electronics Engineers (IEEE), S. 1-12. DOI: 10.23919/SpringSim.2019.8732871.

Albus, J. & Antsaklis, P. J. (1998): Autonomy in Engineering Systems: What is it and Why is it Important? Setting the Stage: Some Autonomous Thoughts on Autonomy. In *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intelligent Systems and Semiotics (ISAS) (Cat. No.98CH36262):* IEEE, S. 520-521. DOI: 10.1109/ISIC.1998.713716.

Alghodhaifi, H. & Lakshmanan, S. (2021): Autonomous Vehicle Evaluation: A Comprehensive Survey on Modeling and Simulation Approaches. *IEEE Access*, 9, S. 151531–151566. DOI: 10.1109/ACCESS.2021.3125620.

Alho, A., Bhavathrathan, B. K., Stinson, M., Gopalakrishnan, R., Le, D.-T. & Ben-Akiva, M. (2017): A multi-scale agent-based modelling framework for urban freight distribution. *Transportation Research Procedia*, 27, S. 188–196. DOI: 10.1016/j.trpro.2017.12.138.

Ali, Z., Biglari, R., Denil, J., Mertens, J., Poursoltan, M. & Traoré, M. K. (2024): From modeling and simulation to Digital Twin: evolution or revolution? *SIMULATION*. DOI: 10.1177/00375497241234680.

Altiok, T. & Melamed, B. (2007): *Simulation modeling and analysis with Arena.* Academic Press. ISBN: 0123705231

American Bureau of Shipping (2020): *ADVISORY ON AUTONOMOUS FUNCTIONALITY* [online]. American Bureau of Shipping. Verfügbar unter: https://ww2.eagle.org/content/dam/eagle/advisories-and-debriefs/abs-advisory-on-autonomous-functionality.pdf [letzter Zugriff: 11. Oktober 2024].

Argüello, L. & Miró, J. (2000): *Distributed Interactive Simulation for Space Projects* [online]. European Space Agency (ESA). Verfügbar unter: https://esa.int/esapub/bulletin/bullet102/Arguello102.pdf [letzter Zugriff: 4. Mai 2023].

Askari, H. R. & Hossain, M. N. (2022): Towards utilizing autonomous ships: A viable advance in industry 4.0. *Journal of International Maritime Safety, Environmental Affairs, and Shipping*, 6 (1), S. 39–49. DOI: 10.1080/25725084.2021.1982637.

Atkinson, C. & Kühne, T. (2003): Model-driven development: a metamodeling foundation. *IEEE Software*, 20 (5), S. 36–41. DOI: 10.1109/MS.2003.1231149.

Atkinson, C. & Kühne, T. (2005): Concepts for Comparing Modeling Tool Architectures. In Hutchison, D. et al. (Hrsg.), *Model Driven Engineering Languages and Systems:* Springer Berlin Heidelberg, S. 398-413. DOI: 10.1007/11557432_30.

Austel, A., Steidel, M. & Westphal, B. (2024): Formal Specification of Traffic Scenarios in Scenario-based Testing of Maritime Assistance Systems [online]. In *4th European Workshop on Maritime Systems, Resilience and Security 2024 (MARESEC 24)*. DOI: 10.5281/ZENODO.14214762.

Awais, M. U., Palensky, P., Elsheikh, A., Widl, E. & Matthias, S. (2013): The high level architecture RTI as a master to the functional mock-up interface components. In *2013 International Conference on Computing, Networking and Communications (ICNC):* IEEE, S. 315-320. DOI: 10.1109/ICCNC.2013.6504102.

Bach, J., Otten, S. & Sax, E. (2016): Model based scenario specification for development and test of automated driving functions. In *2016 IEEE Intelligent Vehicles Symposium (IV):* IEEE, S. 1149-1155. DOI: 10.1109/IVS.2016.7535534.

Bagschik, G., Menzel, T. & Maurer, M. (2018): Ontology based Scene Creation for the Development of Automated Vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV):* IEEE, S. 1813-1820. DOI: 10.1109/IVS.2018.8500632.

Bak, P. M. (2019): *Lernen, Motivation und Emotion - Allgemeine Psychologie II - das Wichtigste, prägnant und anwendungsorientiert.* Springer Berlin Heidelberg: Angewandte Psychologie Kompakt. DOI: 10.1007/978-3-662-59691-3.

Balci, O., Ball, G. L., Morse, K. L., Page, E., Petty, M. D., Tolk, A. & Veautour, S. N. (2017): Model Reuse, Composition, and Adaptation. In Fujimoto, R. et al. (Hrsg.), *Research Challenges in Modeling and Simulation for Engineering Complex Systems:* Springer International Publishing, S. 87-115. DOI: 10.1007/978-3-319-58544-4_6.

Balzert, H. (2011): *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb.* Spektrum Akademischer Verlag. DOI: 10.1007/978-3-8274-2246-0.

Balzert, H., Schröder, M. & Schäfer, C. (2022): *Wissenschaftliches Arbeiten - Ethik, Inhalt & Form wiss. Arbeiten, Handwerkszeug, Quellen, Projektmanagement, Präsentation*.

Banks, J. et al. (2005): *Discrete-Event System Simulation* (4. Aufl.). Prentice Hall: Prentice Hall International Series in Industrial & Systems Engineering. ISBN: 0131293427

Banzal, S. (2020): *XML basics.* Mercury Learning and Information. ISBN: 9781683925460

Barceló, J. (2010a): Models, Traffic Models, Simulation, and Traffic Simulation. In Barceló, J. (Hrsg.), *Fundamentals of Traffic Simulation:* Springer New York, S. 1-62. DOI: 10.1007/978-1-4419-6142-6_1.

Barceló, J. (2010b): Preface. In Barceló, J. (Hrsg.), *Fundamentals of Traffic Simulation:* Springer New York, S. vii-viii. ISBN: 978-1-4419-6141-9.

Barker, S. (2018): *Aircraft as a System-of-Systems - A business process approach for the aerospace industry.* Society of Automotive Engineers: Society of Automotive Engineers. Electronic publications. ISBN: 9780768095487

Baugehn, S. & Tettenborn, A. (2021): International Regulation of Shipping and Unmanned Vessels. In Soyer, B. & Tettenborn, A. (Hrsg.), *Artificial intelligence and autonomous shipping. Developing the international legal framework:* Hart Publishing, an imprint of Bloomsbury Publishing, S. 7-24. ISBN: 9781509933358.

Ben Abdessalem, R., Nejati, S., Briand, L. C. & Stifter, T. (2016): Testing advanced driver assistance systems using multi-objective search and neural networks. In Lo, D., Apel, S. & Khurshid, S. (Hrsg.), *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering:* ACM, S. 63-74. DOI: 10.1145/2970276.2970311.

Bentaleb, O., Belloum, A. S., Sebaa, A. & El-Maouhab, A. (2022): Containerization technologies: taxonomies, applications and challenges. *The Journal of Supercomputing.* Springer Nature, 78 (1), S. 1144–1181. DOI: 10.1007/s11227-021-03914-1.

Bérard, B. et al. (2001): *Systems and Software Verification.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-04558-9.

Bill, R. (2016): *Grundlagen der Geo-Informationssysteme* (6., völlig neu bearbeitete und erweiterte Auflage). Wichmann. ISBN: 978-3-87907-608-6

Blache, H., Laharotte, P.-A. & El Faouzi, N.-E. (2023): Towards an Effective Generation of Functional Scenarios for AVs to Guide Sampling. In Guiochet, J. et al. (Hrsg.), *Computer Safety, Reliability, and Security. SAFECOMP 2023 Workshops:* Springer Nature Switzerland, S. 260-270. DOI: 10.1007/978-3-031-40953-0_22.

Blochwitz, 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. In *Proceedings of the 9th International MODELICA Conference, September 3-5, 2012, Munich, Germany:* Linköping University Electronic Press, S. 173-184. DOI: 10.3384/ecp12076173.

Bocciarelli, P., D'Ambrogio, A., Falcone, A., Garro, A. & Giglio, A. (2016): A Model-Driven Approach to Enable the Distributed Simulation of Complex Systems. In Auvray, G. et al. (Hrsg.), *Complex Systems Design & Management:* Springer International Publishing, S. 171-183. DOI: 10.1007/978-3-319-26109-6_13.

Bocciarelli, P., D'Ambrogio, A. & Fabiani, G. (2012): A Model-driven Approach to Build HLA-based Distributed Simulations from SysML Models. In *Proceedings of the 2nd International Conference on Simulation and Modeling Methodologies, Technologies and Applications:* SciTePress - Science and and Technology Publications, S. 49-60. DOI: 10.5220/0004059900490060.

Bocciarelli, P., D'Ambrogio, A., Giglio, A. & Gianni, D. (2013): A SaaS-based automated framework to build and execute distributed simulations from SysML models. In *2013 Winter Simulations Conference (WSC):* IEEE, S. 1371-1382. DOI: 10.1109/WSC.2013.6721523.

Bocciarelli, P., D'Ambrogio, A., Giglio, A. & Paglia, E. (2019): Automated Generation of FOM Modules for HLA-Based Distributed Simulations. In *2019 Spring Simulation Conference (SpringSim):* IEEE, S. 1-12. DOI: 10.23919/SpringSim.2019.8732865.

Bocciarelli, P., D'Ambrogio, A., Giglio, A. & Paglia, E. (2020): Model-Driven Distributed Simulation Engineering. In Mustafee, N. (Hrsg.), *2019 Winter Simulation Conference (WSC):* IEEE, S. 75-89. DOI: 10.1109/WSC40007.2019.9004937.

Bock, J., Krajewski, R., Eckstein, L., Klimke, J., Sauerbier, J. & Zlocki, A. (2018): Data basis for scenario-based validation of HAD on highways. In Eckstein, L. et al. (Hrsg.), *27th Aachen colloquium automobile and engine technology:* Institute for Automotive Engineering, RWTH Aachen, S. 8-10. ISBN: 978-3-00-057468-9.

Bononi, L., Bracuto, M., D'Angelo, G. & Donatiello, L. (2005): Scalable and Efficient Parallel and Distributed Simulation of Complex, Dynamic and Mobile Systems. In *2005 Workshop on Techniques, Methodologies and Tools for Performance Evaluation of Complex Systems (FIRB-PERF'05):* IEEE, S. 136-145. DOI: 10.1109/FIRB-PERF.2005.17.

Borgeest, K. (2023): *Elektronik in der Fahrzeugtechnik.* Springer Fachmedien Wiesbaden. DOI: 10.1007/978-3-658-41483-2.

Bortz, J. & Döring, N. (2002): *Forschungsmethoden und Evaluation - für Human- und Sozialwissenschaftler* (3. Aufl.). Springer Berlin, Heidelberg: Springer-Lehrbuch. DOI: 10.1007/978-3-662-07299-8.

Bortz, J. & Schuster, C. (2010): *Statistik für Human- und Sozialwissenschaftler* (7., vollständig überarbeitete und erweiterte Auflage). Springer-Verlag Berlin Heidelberg: Springer-Lehrbuch. DOI: 10.1007/978-3-642-12770-0.

Bortz, J. (1979): *Lehrbuch der Statistik.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-08343-7.

Bossel, H. (1994): *Modeling and simulation.* CRC Press. ISBN: 9781439863565

Bossel, H. (2004): *Systeme, Dynamik, Simulation - Modellbildung, Analyse und Simulation komplexer Systeme.* Books on Demand. ISBN: 9783833409844

Boudardara, F., Boussif, A., Meyer, P.-J. & Ghazel, M. (2023): A review of abstraction methods towards verifying neural networks. *ACM Transactions on Embedded Computing Systems*. DOI: 10.1145/3617508.

Bräuninger, M., Fiedler, R., Friedrich, T., Küchle, J., Maatsch, S., Schlennstedt, J., Stiller, S. & Teuber, M.-O. (2021): *Volkswirtschaftliche Bedeutung des Hamburger Hafens - Untersuchung der regional- und gesamtwirtschaftlichen Bedeutung des Hamburger Hafens* [online] (Final Report). Institute of Shipping Economics and Logistics. Verfügbar unter: https://hamburg-port-authority.de/fileadmin/user_upload/BeschaeftigungsstudieHafenHamburg2019_Endbericht_final.pdf [letzter Zugriff: 1. April 2021].

Brinkmann, M. & Hahn, A. (2017): Testbed architecture for maritime cyber physical systems. In *IEEE 15th International Conference on Industrial Informatics (INDIN):* IEEE, S. 923-928. DOI: 10.1109/INDIN.2017.8104895.

Brinkmann, M. (2018): *Physikalische Testfeld-Architektur für die Unterstützung der Entwicklung von automatisierten Schiffsführungssystemen.* Dissertation.

Brinkmann, M., Bode, E., Lamm, A., Maelen, S. V. & Hahn, A. (2017): Learning from automotive: Testing maritime assistance systems up to autonomous vessels. In *OCEANS 2017 - Aberdeen:* IEEE, S. 1-8. DOI: 10.1109/OCEANSE.2017.8084951.

Brinkmann, M., Hahn, A. & Abdelaal, M. (2018): Vessel-in-the-Loop Architecture for Testing Highly Automated Maritime Systems. In Bertram, V. (Hrsg.), *COMPIT' 18. 17th International Conference on Computer and IT Applications in the Maritime Industries : Pavone, 14-16 May 2018:* Technische Universität Hamburg-Harburg. ISBN: 978-3-89220-707-8.

Brooks, F. (1987): No Silver Bullet Essence and Accidents of Software Engineering. *Computer.* IEEE, 20 (4), S. 10–19. DOI: 10.1109/MC.1987.1663532.

Bücs, R. L., Murillo, L. G., Korotcenko, E., Dugge, G., Leupers, R., Ascheid, G., Ropers, A., Wedler, M. & Hoffmann, A. (2015): Virtual Hardware-In-The-Loop Co-Simulation for Multi-Domain Automotive Systems via the Functional Mock-Up Interface. In *2015 Forum on Specification and Design Languages (FDL):* IEEE, S. 1-8. DOI: 10.1109/FDL.2015.7306355.

Budde, Kautz, Kuhlenkamp & Züllighoven (Hrsg.) (1992): *Prototyping - An Approach to Evolutionary System Development.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-76820-0.

Bundesministerium für Wirtschaft und Energie (BMWi) (2017): *Maritime Agenda 2025. Für die Zukunft des maritimen Wirtschaftsstandortes Deutschland.* [online]. Verfügbar unter: https://bmwk.de/Redaktion/DE/Publikationen/Wirtschaft/maritime-agenda-2025.pdf [letzter Zugriff: 28. November 2022].

Bundesministerium für Wirtschaft und Energie (BMWi) (2018): *Maritime Forschungsstrategie 2025* [online]. Verfügbar unter: https://bmwk.de/Redaktion/DE/Publikationen/Technologie/maritime-forschungsstrategie-2025.html [letzter Zugriff: 28. November 2022].

Bungartz, H.-J. (2013): *Modellbildung und Simulation - Eine anwendungsorientierte Einführung* (2., überarb. Aufl. 2013). Springer Spektrum: SpringerLink Bücher. DOI: 10.1007/978-3-642-37656-6.

Bureau Veritas (2019): *Guidelines for Autonomous Shipping - NI 641 DT R01 E* [online]. Bureau Veritas. Verfügbar unter: https://erules.veristar.com/dy/data/bv/pdf/641-NI_2019-10.pdf [letzter Zugriff: 10. August 2023].

Burger, M., van den Berg, M., Hegyi, A., Schutter, B. de & Hellendoorn, J. (2013): Considerations for model-based traffic control. *Transportation Research Part C: Emerging Technologies*, 35, S. 1–19. DOI: 10.1016/j.trc.2013.05.011.

Burmeister, H, Bruhn, W, Rødseth & Ø (2014): Can unmanned ships improve navigational safety? In *Transport Research Arena 2014 (TRA2014).*

Carpin, S., Lewis, M., Wang, J., Balakirsky, S. & Scrapper, C. (2007): USARSim: a robot simulator for research and education. In *Proceedings of the IEEE International Conference on Robotics and Automation:* IEEE, S. 1400-1405. DOI: 10.1109/ROBOT.2007.363180.

Cavacini, A. (2015): What is the best database for computer science journal articles? *Scientometrics*, 102 (3), S. 2059–2071. DOI: 10.1007/s11192-014-1506-1.

Cho, D.-S., Yun, S., Kim, H., Kwon, J. & Kim, W.-T. (2020): Autonomous Driving System Verification Framework with FMI Co-Simulation based on OMG DDS. In *2020 IEEE International Conference on Consumer Electronics (ICCE):* IEEE, S. 1-6. DOI: 10.1109/ICCE46568.2020.9043010.

Chwif, L., Barretto, M. & Paul, R. J. (2000): On simulation model complexity. In *2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165):* IEEE, S. 449-455. DOI: 10.1109/WSC.2000.899751.

Cleff, T. (2015): *Deskriptive Statistik und Explorative Datenanalyse.* Gabler Verlag. DOI: 10.1007/978-3-8349-4748-2.

Cooper, H. M. (1988): Organizing knowledge syntheses: A taxonomy of literature reviews. *Knowledge in Society*, 1 (1), S. 104–126.

Costa, A., Innamaa, S. & Malin, F. (2025): Safety impact of advanced driver assistance systems in Europe in 2030. *Journal of Safety Research*, 95, S. 290–300. DOI: 10.1016/j.jsr.2025.10.008.

Da Bruto Costa, A. A., Irvine, P., Dodoiu, T., Khastgir, S. & Jennings, P. (2025): Building a Robust Scenario Library for Safety Assurance of Automated Driving Systems: A Review. *IEEE Access*, S. 1. DOI: 10.1109/ACCESS.2025.3584206.

Dahanayake, A. & Thalheim, B. (2010): Co-evolution of (Information) System Models. In Bider, I. et al. (Hrsg.), *Enterprise, Business-Process and Information Systems Modeling. 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010, Hammamet, Tunisia. Proceedings:* Springer-Verlag Berlin Heidelberg, S. 314-326. ISBN: 978-3-642-13051-9.

Dählmann, K. (2022): *Simulation von Aktzeptanz und Nutzung von Mobilitätsangeboten.* Dissertation.

Dahmann, J. S. (2015a): *Systems of Systems Characterization and Types* [online] (Systems of Systems Engineering for NATO Defence Applications (STO-EN-SCI-276)). North Atlantic Treaty Organization (NATO): STO EDUCATIONAL NOTES, (01). Verfügbar unter: https://sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-SCI-276/EN-SCI-276-01.pdf [letzter Zugriff: 10. Januar 2024].

Dahmann, J. S. (2015b): *Systems of Systems Engineering Life Cycle* [online] (Systems of Systems Engineering for NATO Defence Applications (STO-EN-SCI-276)). North Atlantic Treaty Organization (NATO): STO EDUCATIONAL NOTES, (04). Verfügbar unter: https://sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-SCI-276/EN-SCI-276-04.pdf [letzter Zugriff: 10. Januar 2024].

Dahmann, J. S., Kuhl, F. & Weatherly, R. (1998): Standards for Simulation: As Simple As Possible But Not Simpler The High Level Architecture For Simulation. *SIMULATION*, 71 (6), S. 378–387. DOI: 10.1177/003754979807100603.

Dai, T., Yu, M. & Wu, Y. (2024): When to announce the queueing information for bounded rationality customers: a discrete-event–based simulation model. *SIMULATION*, 100 (10), S. 997–1017. DOI: 10.1177/00375497241236968.

D'Ambrogio, A., Bocciarelli, P., Delfa, J. & Kisdi, A. (2020): Application of a Model-driven Approach to the Development of Distributed Simulations: The Esa Hraf Case. In *Spring Simulation Conference (SpringSim 2020):* Society for Modeling and Simulation International (SCS). DOI: 10.22360/SpringSim.2020.Mod4Sim.002.

Damm, W., 2018: *Traffic Sequence Charts - A Formal Visual Specification Language for Requirement Capture and Specification Development of Highly Autonomous Cars* [online], 2018. Verfügbar unter: https://dlr.de/fs/Portaldata/16/Resources/veranstaltungen/2018/Vortrag_WD_DLR_Workshop_Testing.pdf [letzter Zugriff: 25. August 2020].

Damsara, K. D. & Barros, A. G. de (2025): A Systematic Review on User Acceptance of Advanced Driver Assistance Systems (ADAS). *Transportation Research Procedia*, 82, S. 3472–3482. DOI: 10.1016/j.trpro.2024.12.082.

Davis, P. V., Dove, M. J. & Stockel, C. T. (1980): A Computer Simulation of Marine Traffic Using Domains and Arenas. *Journal of Navigation*, 33 (2), S. 215–222. DOI: 10.1017/S0373463300035220.

Denil, J. (2023): Validity in (Co-) Simulation. In Masci, P. et al. (Hrsg.), *Software Engineering and Formal Methods. SEFM 2022 Collocated Workshops:* Springer International Publishing, S. 193-199. DOI: 10.1007/978-3-031-26236-4_17.

Department of Defense, Modeling and Simulation Coordination Office (2011): *Modeling and Simulation (M&S) Glossary*.

Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR) (2019): *Pegasus Method - An Overview* [online]. Federal Ministry for Economic Affairs and Energy. Verfügbar unter: https://pegasusprojekt.de/files/tmpl/Pegasus-Abschlussveranstaltung/PEGASUS-Gesamtmethode.pdf [letzter Zugriff: 10. Dezember 2020].

Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR) (2022). *SET Level Projekt* [online]. Verfügbar unter: https://setlevel.de/ [letzter Zugriff: 19. Februar 2024].

Dibbern, C. & Hahn, A. (2014): Maritime Traffic Co-Simulation for Analyses of Maritime Systems [online]. In *28th European Conference on Modelling and Simulation ECMS 2014:* Modeling & Simulation Center - Laboratory of Enteprise Solutions (MSC-LES).

Dierßen, S. (2002): *Systemkopplung zur komponentenorientierten Simulation digitaler Produkte.* VDI Verlag: Fortschritt-Berichte VDI. Reihe 20, Rechnerunterstützte Verfahren. Nr. 358. ISBN: 3-18-335820-4

Dijkstra, E. W. (1971): Hierarchical ordering of sequential processes. *Acta Informatica*, 1 (2), S. 115–138. DOI: 10.1007/BF00289519.

Ding, W., Chen, B., Xu, M. & Zhao, D. (2021): Learning to Collide: An Adaptive Safety-Critical Scenarios Generating Method. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS):* IEEE, S. 2243-2250. DOI: 10.1109/IROS45743.2020.9340696.

Dirnfeld, R., Donato, L. de, Somma, A., Azari, M. S., Marrone, S., Flammini, F. & Vittorini, V. (2024): Integrating AI and DTs: challenges and opportunities in railway maintenance application and beyond. *SIMULATION*, 100 (9), S. 903–917. DOI: 10.1177/00375497241229756.

Dittmann, K. (2022): Autonomy for Ships: System Thinking and Engineering. In *Proceedings of International Conference on Software, Telecommunications and Computer Networks 2022:* IEEE, S. 1-6. DOI: 10.23919/SoftCOM55329.2022.9911446.

DNV, CG-0264:2021. DNV SE (2021)**:** *Autonomous and remotely operated ships*.

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A. & Koltun, V. (2017): CARLA: An Open Urban Driving Simulator [online]. In Levine, S., Vanhoucke, V. & Goldberg, K. (Hrsg.), *Proceedings of the 1st Annual Conference on Robot Learning:* PMLR, S. 1-16.

Drechsler, M. F., Sharma, V., Reway, F., Schütz, C. & Huber, W. (2022): Dynamic Vehicle-in-the-Loop: A Novel Method for Testing Automated Driving Functions. *SAE International Journal of Connected and Automated Vehicles*, 5 (4). DOI: 10.4271/12-05-04-0029.

Dreossi, T., Fremont, D. J., Ghosh, S., Kim, E., Ravanbakhsh, H., Vazquez-Chanlatte, M. & Seshia, S. A. (2019): VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In Dillig, I. & Tasiran, S. (Hrsg.), *Computer Aided Verification:* Springer International Publishing, S. 432-442. DOI: 10.1007/978-3-030-25540-4_25.

Drogoul, A., Vanbergue, D. & Meurisse, T. (2003): Multi-agent Based Simulation: Where Are the Agents? In Goos, G. et al. (Hrsg.), *Multi-Agent-Based Simulation II:* Springer Berlin Heidelberg, S. 1-15. DOI: 10.1007/3-540-36483-8_1.

Durlik, I., Miller, T., Kostecka, E., Kozlovska, P. & Ślączka, W. (2025): Enhancing Safety in Autonomous Maritime Transportation Systems with Real-Time AI Agents. *Applied Sciences*, 15 (9), S. 4986. DOI: 10.3390/app15094986.

Düser, T. (2010): *X-in-the-Loop - ein durchgängiges Validierungsframework für die Fahrzeugentwicklung am Beispiel von Antriebsstrangfunktionen und Fahrerassistenzsystemen. X-in-the-Loop - an integrated validation framework for vehicle development using powertrain functions and driver assistance systems.* Dissertation. Karlsruher Institut für Technologie (KIT): Forschungsberichte. IPEK. 47. DOI: 10.5445/IR/1000020671.

Elhadeedy, A. & Daily, J. (2024): Autonomous Vehicle Development as a System-of-Systems and Constituent Systems Networking. In *2024 IEEE International Systems Conference (SysCon):* IEEE, S. 1-7. DOI: 10.1109/SysCon61195.2024.10553603.

Elspas, P., Langner, J., Aydinbas, M., Bach, J. & Sax, E. (2020): Leveraging Regular Expressions for Flexible Scenario Detection in Recorded Driving Data. In *2020 IEEE International Symposium on Systems Engineering (ISSE):* IEEE, S. 1-8. DOI: 10.1109/ISSE49799.2020.9272025.

Ernst, H., Schmidt, J. & Beneken, G. (2016): Höhere Programmiersprachen und C. In Ernst, H., Schmidt, J. & Beneken, G. (Hrsg.), *Grundkurs Informatik:* Springer Fachmedien Wiesbaden, S. 597-665. DOI: 10.1007/978-3-658-14634-4_14.

ESA, SMP2. European Space Agency (ESA) (2005)**:** *SMP 2.0 Handbook - EGOS-SIM-GEN-TN-0099.* Verfügbar unter: https://taste.tuxfamily.org/wiki/images/9/9a/SMP_2.0_Handbook_-_1.2.pdf [letzter Zugriff: 2. April 2025].

Eschenbach, W. J. von (2021): Transparency and the Black Box Problem: Why We Do Not Trust AI. *Philosophy & Technology*, 34 (4), S. 1607–1622. DOI: 10.1007/s13347-021-00477-0.

Evora Gomez, J., Hernández Cabrera, J. J. & Tavella, J.-P. (2016): Semantic interoperability in co-simulation: use case and requirements [online]. In Evora-Gomez, J. & Hernández Cabrera, J.J. (Hrsg.), *The European Simulation Modelling Conference 2016 (ESM 2016):* The European Technology Institute (ETI). ISBN: 978-90-77381-95-3.

Évora Gómez, J., Hernández Cabrera, J. J., Tavella, J.-P., Vialle, S., Kremers, E. & Frayssinet, L. (2019): Daccosim NG: co-simulation made simpler and faster. In *Proceedings of the 13th International Modelica Conference, Regensburg, Germany, March 4–6, 2019:* Linköing University Electronic Press, S. 785-794. DOI: 10.3384/ecp19157785.

Fan, Y. & Krishnan, K. (2020): *Open Standards - Essential for Self-Driving? - Validating autonomous vehicles through billions of virtual test miles* [online]. Hexagon AB. Verfügbar unter: https://hexagon.com/resources/resource-library/forms/open-standards-essential-for-self-driving-white-paper-rll3-940 [letzter Zugriff: 27. August 2025].

Faÿ, G., González-Arévalo, B., Mikosch, T. & Samorodnitsky, G. (2006): Modeling teletraffic arrivals by a Poisson cluster process. *Queueing Systems*, 54 (2), S. 121–140. DOI: 10.1007/s11134-006-9348-z.

Ferstl, O. K. & Sinz, E. J. (2013): *Grundlagen der Wirtschaftsinformatik* (7., aktualisierte Aufl.). Oldenbourg. ISBN: 9783486713534

Fisahn, A. (2018): Umweltschutz. In Voigt, R. (Hrsg.), *Handbuch Staat:* Springer Fachmedien Wiesbaden, S. 1591-1602. DOI: 10.1007/978-3-658-20744-1_143.

Flood, R. L. & Carson, E. R. (1988): *Dealing with Complexity - An Introduction to the Theory and Application of Systems Science.* Plenum Press. DOI: 10.1007/978-1-4684-7799-3.

Formela, K., Neumann, T. & Weintrit, A. (2019): Overview of Definitions of Maritime Safety, Safety at Sea, Navigational Safety and Safety in General. *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, 13 (2), S. 285–290. DOI: 10.12716/1001.13.02.03.

Forrester, J. W. (1972): *Grundzüge einer Systemtheorie - Principles of Systems.* Gabler Verlag. DOI: 10.1007/978-3-663-02094-3.

Fossen, T. I. (2011): *Handbook of marine craft hydrodynamics and motion control - Vademecum de navium motu contra aquas et de motu gubernando.* Wiley. ISBN: 9781119991496

Fowler, M. (2010): *Domain-specific languages.* Addison Wesley: The Addison-Wesley signature series. ISBN: 9786612797309

Frank, U. (2014): Multilevel Modeling. *Business & Information Systems Engineering*, 6 (6), S. 319–337. DOI: 10.1007/s12599-014-0350-4.

Fremont, D. J., Kim, E., Pant, Y. V., Seshia, S. A., Acharya, A., Bruso, X., Wells, P., Lemke, S., Lu, Q. & Mehta, S. (2020): Formal Scenario-Based Testing of Autonomous Vehicles: From Simulation to the Real World. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC):* IEEE. DOI: 10.1109/ITSC45102.2020.9294368.

Fujimoto, R. & Loper, M. (2017): Introduction. In Fujimoto, R. et al. (Hrsg.), *Research Challenges in Modeling and Simulation for Engineering Complex Systems:* Springer International Publishing. DOI: 10.1007/978-3-319-58544-4_1.

Fujimoto, R. (2015): Parallel and Distributed Simulation. In Yilmaz, L. et al. (Hrsg.), *WSC '15: Proceedings of the 2015 Winter Simulation Conference:* IEEE Press. ISBN: 9781467397414.

Fujimoto, R. M. (1998): Time Management in The High Level Architecture. *SIMULATION*, 71 (6), S. 388–400. DOI: 10.1177/003754979807100604.

Fujimoto, R. M. (2003): Distributed simulation systems. In *Proceedings of the 2003 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.03EX693):* IEEE, S. 124-134. DOI: 10.1109/WSC.2003.1261415.

Fujimoto, R. M. (2016): Research Challenges in Parallel and Distributed Simulation. *ACM Transactions on Modeling and Computer Simulation*, 26 (4), S. 1–29. DOI: 10.1145/2866577.

Gambi, A., Huynh, T. & Fraser, G. (2019): Generating effective test cases for self-driving cars from police reports. In Dumas, M. et al. (Hrsg.), *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering:* ACM, S. 257-267. DOI: 10.1145/3338906.3338942.

Gambi, A., Maul, P., Mueller, M., Stamatogiannakis, L., Fischer, T. & Panichella, S. (2020): *Soft-body Simulation and Procedural Generation for the Development and Testing of Cyber-physical Systems* [online]. BeamNG GmbH. Verfügbar unter: https://beamng.tech/research/ [letzter Zugriff: 22. August 2024].

Gasevic, D., Djuric, D. & Devedzic, V. (2009): *Model Driven Engineering and Ontology Development* (2. Aufl.). Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-00282-3.

Geiger, A., Lenz, P. & Urtasun, R. (2012): Are we ready for autonomous driving? The KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition:* IEEE, S. 3354-3361. DOI: 10.1109/CVPR.2012.6248074.

Geimer, M., Krüger, T. & Linsel. P. (2006): Co-Simulation, gekoppelte Simulation oder Simulatorkopplung?: Ein Versuch der Begriffsvereinheitlichung. *O+P Ölhydraulik und Pneumatik*. GBI-Genios, 50 (11-12), S. 572–576. DOI: 10.5445/IR/1000011318.

Gelder, E. de & Op den Camp, O. (2021): Procedure for the Safety Assessment of an Autonomous Vehicle Using Real-World Scenarios. In *38th FISITA World Congress:* FISITA (UK) Ltd. DOI: 10.46720/f2020-ves-014.

Geyer, S., Baltzer, M., Franz, B., Hakuli, S., Kauer, M., Kienle, M., Meier, S., Weißgerber, T., Bengler, K., Bruder, R., Flemisch, F. & Winner, H. (2014): Concept and development of a unified ontology for generating test and use-case catalogues for assisted and automated vehicle guidance. *IET Intelligent Transport Systems*, 8 (3), S. 183–189. DOI: 10.1049/iet-its.2012.0188.

Ghadai, P., Shree, L. P., Chhatria, L. & Prasad, R. (2016): A study on agent based modelling for traffic simulation. *International Journal of Computer Science and Information Technologies*, 7 (2), S. 932–936.

Giering, J.-E. & Dyck, A. (2021): Maritime Digital Twin architecture. *at - Automatisierungstechnik*, 69 (12), S. 1081–1095. DOI: 10.1515/auto-2021-0082.

Gipser, M. (1999): *Systemdynamik und Simulation.* Vieweg+Teubner Verlag. DOI: 10.1007/978-3-663-11581-6.

Glas, B., Gebauer, C., Hänger, J., Heyl, A., Klarmann, J., Kriso, S., Vembar, P. & Wörz, P. (2015): Automotive safety and security integration challenges. In Klenk, H. et al. (Hrsg.), *GI-Edition Proceedings Band 240 - Automotive - Safety & Security 2015:* Gesellschaft für Informatik e. V., S. 13-28. ISBN: 978-3-88579-634-3.

Go, K. & Carroll, J. M. (2004): The Blind Men and the Elephant: Views of Scenario-Based System Design. *Interactions.* Association for Computing Machinery (ACM), 11 (6), S. 44–53. DOI: 10.1145/1029036.1029037.

Goldsman, D., Nance, R. E. & Wilson, J. R. (2010): A brief history of simulation revisited. In *Proceedings of the 2010 Winter Simulation Conference (WSC). 5 - 8 Dec. 2010, Baltimore, MD, USA ; [incorporating the] MASM (Modeling and Analysis for Semiconductor Manufacturing) Conference:* IEEE, S. 567-574. DOI: 10.1109/WSC.2010.5679129.

Gomes, C., Thule, C., Broman, D., Larsen, P. G. & Vangheluwe, H. (2019): Co-Simulation: A Survey. *ACM Computing Surveys*, 51 (3), S. 1–33. DOI: 10.1145/3179993.

Gräßler, I. & Oleff, C. (2022): *Systems Engineering.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-64517-8.

Graubohm, R., Salem, N. F., Nolte, M. & Maurer, M. (2023): On Assumptions with Respect to Occlusions in Urban Environments for Automated Vehicle Speed Decisions. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC):* IEEE, S. 738-745. DOI: 10.1109/ITSC57777.2023.10422457.

Gronau, N., Kern, E.-M. & Lukas, U. (2003): Integration des Community-Gedankens in das Collaborative Engineering am Beispiel des Schiffbaus. In Uhr, W., Esswein, W. & Schoop, E. (Hrsg.), *Wirtschaftsinformatik 2003 Band II:* Physica-Verlag HD, S. 21-40. DOI: 10.1007/978-3-642-57445-0_2.

Gütlein, M. (2023): *Data-Centric Distributed Simulation in the Traffic Domain* [online]. Dissertation. Verfügbar unter: https://opus4.kobv.de/opus4-fau/frontdoor/index/index/docId/23916 [letzter Zugriff: 15. November 2023].

Gütlein, M., Baron, W., Renner, C. & Djanatliev, A. (2020): Performance Evaluation of HLA RTI Implementations. In *2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT):* IEEE, S. 1-8. DOI: 10.1109/DS-RT50469.2020.9213641.

Hahn, A. & Noack, T. (2016): eMaritime Integrated Reference Platform [online]. In *Deutscher Luft- und Raumfahrtkongress 2016:* Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e. V.

Hahn, A., Schweigert, S., Gollücke, V. & Buschmann, C. (2015): Virtual Testbed for maritime safety assessment. In Maritime University of Szczeci (Hrsg.), (Hrsg.), *The 16th Marine Traffic Engineering Conference and International Symposium Information on Ships MTE-ISIS 2015:* Scientific Journals of the Maritime University of Szczecin, S. 116-122. DOI: 10.17402/065.

Hake, G., Maelen, S. V. & Hahn, A. (2021): Maintaining safety requirements of updated maritime surveillance systems. In Hahn, A. (Hrsg.), *13th IFAC Conference On Control Applications in Marine Systems, Robotics, and Vehicles. Proceedings:* International Federation of Automatic Control (IFAC), S. 112-119. DOI: 10.1016/j.ifacol.2021.10.081.

Hake, G., Reiher, D., Mentjes, J. & Hahn, A. (2024): Integrating scenario- and contract-based verification for automated vessels. *Journal of Marine Science and Technology*. DOI: 10.1007/s00773-024-01008-0.

Halevi, G., Moed, H. & Bar-Ilan, J. (2017): Suitability of Google Scholar as a source of scientific information and as a source of data for scientific evaluation—Review of the Literature. *Journal of Informetrics*, 11 (3), S. 823–834. DOI: 10.1016/j.joi.2017.06.005.

Hanappi, G. (2017): Agent-based modelling. History, essence, future. *PSL Quarterly Review.* PSL Quarterly Review, 70 (283), S. 449–472. DOI: 10.13133/2037-3643_70.283_3.

Hanke, T. (2020): *Virtual Sensorics: Simulated Environmental Perception for Automated Driving Systems* [online]. Dissertation. Verfügbar unter: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20200529-1519952-1-7 [letzter Zugriff: 1. April 2021].

Hansen, S. T., Gomes, C. Â., Najafi, M., Sommer, T., Blesken, M., Zacharias, I., Kotte, O., Mai, P. R., Schuch, K., Wernersson, K., Bertsch, C., Blochwitz, T. & Junghanns, A. (2022): The FMI 3.0 Standard Interface for Clocked and Scheduled Simulations. *Electronics*, 11 (21), S. 3635. DOI: 10.3390/electronics11213635.

Hansen, S. T., Thule, C., Gomes, C., Lausdahl, K. G., Madsen, F. P., Abbiati, G. & Larsen, P. G. (2024): Co-simulation at different levels of expertise with Maestro2. *Journal of Systems and Software*, 209, S. 111905. DOI: 10.1016/j.jss.2023.111905.

Harder, N., Weidlich, A. & Staudt, P. (2023): Modeling Participation of Storage Units in Electricity Markets using Multi-Agent Deep Reinforcement Learning. In *Proceedings of the 14th ACM International Conference on Future Energy Systems:* ACM, S. 439-445. DOI: 10.1145/3575813.3597351.

Harvey, C. & Stanton, N. A. (2014): Safety in System-of-Systems: Ten key challenges. *Safety Science*, 70, S. 358–366. DOI: 10.1016/j.ssci.2014.07.009.

Hasselhorn, M. & Grube, D. (2003): Das Arbeitsgedächtnis: Funktionsweise, Entwicklung und Bedeutung für kognitive Leistungsstörungen. *Sprache · Stimme · Gehör*, 27 (01), S. 31–37. DOI: 10.1055/s-2003-37875.

Hatledal, L. I. (2021): *Protocols and Standard for Simulation Co-Simulation – For demanding maritime operations* [online]. doctoral. Verfügbar unter: https://hdl.handle.net/11250/2755142 [letzter Zugriff: 09/10/23].

Hatledal, L. I., Styve, A., Hovland, G. & Zhang, H. (2019): A Language and Platform Independent Co-Simulation Framework Based on the Functional Mock-Up Interface. *IEEE Access*, 7, S. 109328–109339. DOI: 10.1109/ACCESS.2019.2933275.

Haw, S.-C., Chew, L.-J., Kusumo, D. S., Naveen, P. & Ng, K.-W. (2023): Mapping of extensible markup language-to-ontology representation for effective data integration. *IAES International Journal of Artificial Intelligence (IJ-AI)*, 12 (1), S. 432. DOI: 10.11591/ijai.v12.i1.pp432-442.

Heesen, B. (2021): *Wissenschaftliches Arbeiten.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-662-62548-4.

Held, T. (2010): Supplier Integration as an Improvement Driver – An Analysis of Some Recent Approaches in the Shipbuilding Industry. In Engelhardt-Nowitzki, C., Nowitzki, O. & Zsifkovits, H. (Hrsg.), *Supply Chain Network Management:* Gabler, S. 369-384. DOI: 10.1007/978-3-8349-6000-9_22.

Henderson-Sellers, B., Clark, T. & Gonzalez-Perez, C. (2013): On the Search for a Level-Agnostic Modelling Language. In Salinesi, C., Norrie, M.C. & Pastor, Ó. (Hrsg.), *Advanced Information Systems Engineering. 25th Internation Conference, CAiSE 2013:* Springer Berlin Heidelberg, S. 240-255. DOI: 10.1007/978-3-642-38709-8_16.

Henshaw, M. (2015): *A Socio-Technical Perspective on SoSE* [online] (Systems of Systems Engineering for NATO Defence Applications (STO-EN-SCI-276)). North Atlantic Treaty Organization (NATO): STO EDUCATIONAL NOTES, (05).

Verfügbar unter: https://sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-SCI-276/EN-SCI-276-05.pdf [letzter Zugriff: 10. Januar 2024].

Herrmann, A. (2022): *Grundlagen der Anforderungsanalyse - Standardkonformes Requirements Engineering.* Springer Fachmedien Wiesbaden. DOI: 10.1007/978-3-658-35460-2.

Hetzel (Hrsg.) (1973): *Program test methods.* Prentice-Hall: Prentice-Hall series in automatic computation. ISBN: 013729624X

Hevner, A. & Chatterjee, S. (2010): *Design Research in Information Systems.* Springer US. 22. DOI: 10.1007/978-1-4419-5653-8.

Hevner, A. R., March, S. T., Park, J. & Ram, S. (2004): Design Science in Information Systems Research. *MIS Quarterly.* Management Information Systems Research Center, University of Minnesota, 28 (1), S. 75–105. Verfügbar unter: http://jstor.org/stable/25148625 [letzter Zugriff: 28. Mai 2024].

Hoppe, I., Hagemann, W., Stierand, I., Hahn, A. & Bolles, A. (2024): Challenges for trustworthy autonomous vehicles: Let us learn from life. *Systems Engineering*, 27 (4), S. 789–800. DOI: 10.1002/sys.21744.

Hui, K. Y., Nguyen, C. H., Lui, G. N. & Liem, R. P. (2023): AirTrafficSim: An open-source web-based air traffic simulation platform. *Journal of Open Source Software (JOSS)*, 8 (86), S. 4916. DOI: 10.21105/joss.04916.

Huiskamp, W. & van den Berg, T. (2016): Federated Simulations. In Setola, R. et al. (Hrsg.), *Managing the Complexity of Critical Infrastructures:* Springer International Publishing, S. 109-137. DOI: 10.1007/978-3-319-51043-9_6.

IEC, 61508:2010. International Electrotechnical Commission (IEC) (2010)**:** *Functional safety of electrical/electronic/programmable electronic safety-related systems.* Verfügbar unter: https://webstore.iec.ch/publication/22273 [letzter Zugriff: 30. Juni 2021].

IEEE, 1516.1:2010. Institute of Electrical and Electronics Engineers (IEEE) (2010)**:** *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification*.

IEEE, 1516.2:2010. Institute of Electrical and Electronics Engineers (IEEE) (2010)**:** *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification*.

IEEE, 1516.3:2003. Institute of Electrical and Electronics Engineers (IEEE) (2003)**:** *Recommended Practice for High Level Architecture (HLA) Federation Development and Execution Process (FEDEP)*.

IEEE, 1516:2010. Institute of Electrical and Electronics Engineers (IEEE) (2010)**:** *Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*.

IEEE, 1730:2022. Institute of Electrical and Electronics Engineers (IEEE) (2022)**:** *Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP)*.

IMO, MSC.1/Circ.1580. International Maritime Organization (IMO) (2017)**:** *Guidelines for Vessels and Units With Dynamic Positioning (DP) Systems*.

International Maritime Organization (IMO) (2018): *Maritime Safety Committee (MSC) - 100th session, 3-7 December 2018* [online]. Verfügbar unter: https://imo.org/en/MediaCentre/MeetingSummaries/Pages/MSC-100th-session.aspx [letzter Zugriff: 15. Dezember 2020].

International Maritime Organization (IMO) (2021a): *Autonomous ships: regulatory scoping exercise completed* [online]. Verfügbar unter: https://imo.org/en/MediaCentre/PressBriefings/pages/MASSRSE2021.aspx [letzter Zugriff: 12. April 2022].

International Maritime Organization (IMO) (2021b): *Outcome of the Regulatory Scoping Exercise for the use of Maritime Autonomous Surface Ships (MASS) - MSC.1/Circ.1638.* International Maritime Organization (IMO).

International Maritime Organization (IMO) (2022). *Maritime Safety Committee (MSC 105), 20-29 April 2022* [online]. Verfügbar unter: https://imo.org/en/MediaCentre/MeetingSummaries/Pages/MSC-105th-session.aspx [letzter Zugriff: 30. August 2024].

Iskandar, R., Dugdale, J., Beck, E. & Cornou, C. (2024): Agent-based simulation of seismic crisis including human behavior: application to the city of Beirut, Lebanon. *SIMULATION*, 100 (4), S. 357–377. DOI: 10.1177/00375497231194608.

ISO, 17894:2005. International Organization for Standardization (ISO) (2005)**:** *Ships and marine technology - Computer applications - General principles for the development and use of programmable electronic systems in marine applications.* Verfügbar unter: https://iso.org/standard/31619.html [letzter Zugriff: 15. April 2021].

ISO, 26262-1:2018. International Organization for Standardization (ISO) (2018)**:** *Road vehicles — Functional safety* . Verfügbar unter: https://iso.org/standard/68383.html [letzter Zugriff: 16. April 2021].

ISO/IEC/IEEE, 15288:2015. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE) (2015)**:** *Systems and software engineering - Systems life cycle processes.* Verfügbar unter: https://iso.org/standard/63711.html [letzter Zugriff: 22. April 2021].

ISO/IEC/IEEE, 24765:2017. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE) (2017)**:** *Systems and software engineering - Vocabulary.* Verfügbar unter: https://ieeexplore.ieee.org/servlet/opac?punumber=8016710 [letzter Zugriff: 3. Juni 2023].

ISO/IEC/IEEE, 29148:2018. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE) (2018)**:** *Systems and software engineering - Life cycle processes — Requirements engineering.*

ISO/IEC/IEEE, 29199-1:2022. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE) (2022)**:** *Software and systems engineering - Software testing - Part 1: General concepts.* Verfügbar unter: https://iso.org/standard/81291.html

ISO/IEC/IEEE, 29199-4:2021. International Organization for Standardization (ISO), International Electrotechnical Commission (IEC), Institute of Electrical and Electronics Engineers (IEEE) (2021)**:** *Software and systems engineering - Software testing - Part 4: Test techniques.* Verfügbar unter: https://iso.org/standard/81291.html

ISO/PAS, 21448:2019. International Organization for Standardization (ISO) (2019)**:** *Road vehicles — Safety of the intended functionality.* Verfügbar unter: https://iso.org/standard/77490.html [letzter Zugriff: 17. Februar 2023].

Jesenski, S., Stellet, J. E., Branz, W. & Zollner, J. M. (2019): Simulation-Based Methods for Validation of Automated Driving: A Model-Based Analysis and an Overview about Methods for Implementation. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC):* IEEE, S. 1914-1921. DOI: 10.1109/ITSC.2019.8917072.

Jorgensen, R. N. (2017). *BIMCO and CIRM propose software maintenance standard for shipping* [online]. Verfügbar unter: https://bimco.org/news/priority-news/20171214_software-maintenance [letzter Zugriff: 31. Oktober 2021].

Junghanns, A., Gomes, C., Schulze, C., Schuch, K., R, P., Blaesken, M., Zacharias, I., Pillekeit, A., Wernersson, K., Sommer, T., Bertsch, C., Blochwitz, T. & Najafi, M. (2021): The Functional Mock-up Interface 3.0 - New Features Enabling New Applications. In Sjölund, M. et al. (Hrsg.), *Proceedings of 14th Modelica Conference 2021, Linköping, Sweden, September 20-24, 2021:* Linköping University Electronic Press, S. 17-26. DOI: 10.3384/ecp2118117.

Kalra, N. & Paddock, S. M. (2016): Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability? *Transportation Research Part A: Policy and Practice*, 94, S. 182–193. DOI: 10.1016/j.tra.2016.09.010.

Kaluza, P., Kölzsch, A., Gastner, M. T. & Blasius, B. (2010): The complex network of global cargo ship movements. *Journal of the Royal Society, Interface*, 7 (48), S. 1093–1103. DOI: 10.1098/rsif.2009.0495.

Karimi, A. & Duggirala, P. S. (2022): Automatic Generation of Test-cases of Increasing Complexity for Autonomous Vehicles at Intersections. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS):* IEEE, S. 1-11. DOI: 10.1109/ICCPS54341.2022.00008.

Karthaus, C., Bick, A., Douglas, B. & Hanke, F. (2021): X-in-the-Loop - Eine durchgängige Erprobungsmethodik. *AT-Zelektronik*, 16 (11), S. 46–51. DOI: 10.1007/s35658-021-0688-6.

Kasereka, S., Zohinga, G., Kiketa, V., Ngoie, R.-B., Mputu, E., Kasoro, N. & Kyandoghere, K. (2023): Equation-Based Modeling vs. Agent-Based Modeling with Applications to the Spread of COVID-19 Outbreak. *Mathematics*, 11 (1), S. 253. DOI: 10.3390/math11010253.

Kay, S., Kisdi, A., Buckley, K., Möller, B., Gray, T., Bocciarelli, P., D'Ambrogio, A., Dankiewicz, I., Kemp, J., Delfa, J. & Ocon, J. (2021): Development of a Distributed Simulation Environment and Model Driven Engineering Framework to Support the Verification & Validation of Complex Autonomy Components. In *International Astronautical Congress – IAC 2021.*

Kearney, J., Willemsen, P., Donikian, S. & Devillers, F. (1999): Scenario languages for driving simulation. *DSC'99.*

Keating, C., Rogers, R., Unal, R., Dryer, D., Sousa-Poza, A., Safford, R., Peterson, W. & Rabadi, G. (2003): System-of-Systems Engineering. *Engineering Management Journal*, 15 (3), S. 36–45. DOI: 10.1080/10429247.2003.11415214.

Kessler, G. C. (2021): The CAN Bus in the Maritime Environment – Technical Overview and Cybersecurity Vulnerabilities. *TransNav, the International Journal on Marine Navigation and Safety of Sea Transportation*, 15 (3), S. 531–540. DOI: 10.12716/1001.15.03.05.

Khabir, M., Emad, G. R., Shahbakhsh, M. & Dulebenets, M. A. (2025): A Strategic Pathway to Green Digital Shipping. *Logistics*, 9 (2), S. 68. DOI: 10.3390/logistics9020068.

Khashe, Y. & Levy, S. (2020): A High Reliability Organization (HRO)-based Retrospective Analysis of Boeing 737 Max Crashes. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 64 (1), S. 851–855. DOI: 10.1177/1071181320641197.

Kidd, M. & Stepan, G. (2014): Delayed control of an elastic beam. *International Journal of Dynamics and Control*, 2 (1), S. 68–76. DOI: 10.1007/s40435-014-0079-4.

Kim, S. & Oviedo-Trespalacios, O. (2025): Disuse of advanced driver assistance systems (ADAS). *Journal of Safety Research.* Elsevier Ltd., 95, S. 180–188. DOI: 10.1016/j.jsr.2025.09.004.

Kim, S.-H. & Whitt, W. (2014): Choosing arrival process models for service systems: Tests of a nonhomogeneous Poisson process. *Naval Research Logistics (NRL)*, 61 (1), S. 66–90. DOI: 10.1002/nav.21568.

King, C., Ries, L., Kober, C., Wohlfahrt, C. & Sax, E. (2019): Automated Function Assessment in Driving Scenarios. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST):* IEEE, S. 414-419. DOI: 10.1109/ICST.2019.00050.

Kirov, G. (2015): An Object-Oriented HLA Simulation Study. *Cybernetics and Information Technologies*, 15 (5), S. 121–130. DOI: 10.1515/cait-2015-0022.

Kitchenham, B. A. & Charters, S. (2007): *Guidelines for performing Systematic Literature Reviews in Software Engineering* [online]. Keele University and Durham University Joint Report& Keele University, (EBSE 2007-001). Verfügbar unter: https://elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf [letzter Zugriff: 29. Januar 2021].

Klauda, M., Kriso, S., Hamann, R. & Schaffert, M. (2012): Automotive safety and security from a supplier's perspective. In Plödereder, E. et al. (Hrsg.), *Automotive - Safety & Security 2012:* Gesellschaft für Informatik e.V, S. 13-22.

Klikovits, S., Gambi, A., Dhungana, D. & Rabiser, R. (2024): Leveraging Software Product Lines for Testing Autonomous Vehicles. In Kehrer, T. et al. (Hrsg.), *Proceedings of the 18th International Working Conference on Variability Modelling of Software-Intensive Systems:* ACM, S. 56-60. DOI: 10.1145/3634713.3634720.

Knauss, E., Pelliccione, P., Heldal, R., Ågren, M., Hellman, S. & Maniette, D. (2016): Continuous Integration Beyond the Team. In Genero, M. et al. (Hrsg.), *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement:* ACM, S. 1-6. DOI: 10.1145/2961111.2962639.

Koelsch, G. (2016): *Requirements writing for system engineering - Project success through realistic requirements.* Apress. DOI: 10.1007/978-1-4842-2099-3.

Koenig, N. & Howard, A. (2004): Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS) 2004:* IEEE, S. 2149-2154. DOI: 10.1109/IROS.2004.1389727.

Kolb, N., Jordan, C., Huber, F. & Pretschner, A. (2022): Automatic Evaluation of Automatically Derived Semantic Scenario Instance Descriptions. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC):* IEEE, S. 1565-1571. DOI: 10.1109/ITSC55140.2022.9922013.

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., Kembhavi, A., Gupta, A. & Farhadi, A. (2022): *AI2-THOR: An Interactive 3D Environment for Visual AI - Preprint* [online], (v4). Verfügbar unter: http://arxiv.org/pdf/1712.05474 [letzter Zugriff: 15/08/24].

Kooij, C. & Hekkenberg, R. (2021): The effect of autonomous systems on the crew size of ships – a case study. *Maritime Policy & Management*, 48 (6), S. 860–876. DOI: 10.1080/03088839.2020.1805645.

Koopman, P. & Wagner, M. (2016): Challenges in Autonomous Vehicle Testing and Validation. *SAE International Journal of Transportation Safety.* SAE International, 4 (1), S. 15–24. DOI: 10.4271/2016-01-0128.

Kopec, D. & Tamang, S. (2007): Failures in complex systems. *ACM SIGCSE Bulletin*, 39 (2), S. 180–184. DOI: 10.1145/1272848.1272905.

Kopetz, H. (2019): *Simplicity is Complex - Foundations of Cyber-Physical System Design.* Springer International Publishing. DOI: 10.1007/978-3-030-20411-2.

Koreimann, D. S. (2000): *Grundlagen der Software-Entwicklung.* DE GRUYTER. DOI: 10.1515/9783486803167.

Koukaki, T. & Tei, A. (2020): Innovation and maritime transport: A systematic review. *Case Studies on Transport Policy*, 8 (3), S. 700–710. DOI: 10.1016/j.cstp.2020.07.009.

Kowalk, W. P. (1996): *System, Modell, Programm - Vom GOTO zur objektorientierten Programmierung.* Spektrum Akad. Verl.: Spektrum Lehrbuch. ISBN: 3-8274-0062-7

Kowalk, W. P. (2003): Planetarium Simulation System Usability and Experiments. In M. H. Hamza (Hrsg.), (Hrsg.), *Proceedings of the IASTED International Conference on Modelling and Simulation (MS 2003), February 24-26, 2003, Palm Springs, California, USA:* IASTED/ACTA Press, S. 509-514.

Krajzewicz, D. (2010): Traffic Simulation with SUMO – Simulation of Urban Mobility. In Barceló, J. (Hrsg.), *Fundamentals of Traffic Simulation:* Springer New York, S. 269-293. DOI: 10.1007/978-1-4419-6142-6_7.

Kramer, U. & Neculau, M. (1998): *Simulationstechnik.* Hanser. ISBN: 978-3446192355

Krammer, M., Fritz, J. & Karner, M. (2015): Model-Based Configuration of Automotive Co-Simulation Scenarios. In *Proceedings of the 48th Annual Simulation Symposium:* Society for Computer Simulation International, S. 155-162. DOI: 10.5555/2876341.2876362.

Kröger, F. (2016): Automated Driving in Its Social, Historical and Cultural Contexts. In Maurer, M. et al. (Hrsg.), *Autonomous Driving:* Springer Berlin Heidelberg, S. 41-68. DOI: 10.1007/978-3-662-48847-8_3.

Kromrey, H. (1994): *Empirische Sozialforschung - Modelle und Methoden der Datenerhebung und Datenauswertung* (6. Aufl.). VS Verlag für Sozialwissenschaften: Uni-Taschenbücher. DOI: 10.1007/978-3-322-96184-6.

Kufoalor, D. K., Johansen, T. A., Brekke, E. F., Hepsø, A. & Trnka, K. (2019): Autonomous maritime collision avoidance: Field verification of autonomous surface vehicle behavior in challenging scenarios. *Journal of Field Robotics*, 23 (6), S. 333. DOI: 10.1002/rob.21919.

Kühnel, C., Leitzke, C. & Hoyer, R. (2009): Evaluation of Floating Car Observer Algorithms using microscopic Traffic Flow Simulation. In *16th ITS World Congress:* Curran Associates, Inc. ISBN: 978-1-61738-589-6.

Kurkowski, S., Camp, T. & Colagrosso, M. (2005): MANET simulation studies. *ACM SIGMOBILE Mobile Computing and Communications Review*, 9 (4), S. 50–61. DOI: 10.1145/1096166.1096174.

Lamm, A. & Hahn, A. (2018): Towards Critical-Scenario Based Testing With Maritime Observation Data. In *2018 OCEANS - MTS/IEEE Kobe Techno-Oceans (OTO):* IEEE. DOI: 10.1109/OCEANSKOBE.2018.8559045.

Lamm, A. (2023): *Identifikation von Beinahekollisionen in maritimen Verkehrsdaten als Ground-Truth für szenariobasiertes Testen* [online]. doctoral. Verfügbar unter: https://oops.uni-oldenburg.de/5860/ [letzter Zugriff: 13. September 2023].

Lee, E. A. (2008): Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC):* IEEE, S. 363-369. DOI: 10.1109/ISORC.2008.25.

Leitner, A., Akkermann, A., Hjøllo, B. Å., Wirts, B., Nickovic, D., Möhlmann, E., Holzer, H., van der Voet, J., Niehaus, J., Sarrazin, M., Zofka, M., Rooker, M., Kubisch, M., Paulweber, M., Siegel, M., Rautila, M., Marko, N., Tummeltshammer, P., Rosenberger, P., Rott, R., Muckenhuber, S., Kalisvaart, Graaf, T. de, D'Hondt, T., Fleck, T. & Slavik, Z. (2019): *ENABLE-S3 - Testing & Validation of Highly Automated Systems* [online]. AVL List GmbH. Verfügbar unter: https://offis.de/offis/publikation/testing-validation-of-highly-automated-systems.html [letzter Zugriff: 20. Februar 2025].

Lemper, B., Maatsch, S., Fiedler, R., Bräuninger, M. & Holocher, K.-H. (2019): *Untersuchung der volkswirtschaftlichen Bedeutung der deutschen See- und Binnenhäfen auf Grundlage ihrer Beschäftigungswirkung* [online]. Federal Ministry of Transport and Digital Infrastructure. Verfügbar unter: https://isl.org/public/studienergebnisse/Beschaeftigungseffekte_BMVI_Endbericht5-Final.pdf [letzter Zugriff: 18. April 2021].

Lenz, B. & Fraedrich, E. (2015): Gesellschaftliche und individuelle Akzeptanz des autonomen Fahrens. In Maurer, M. et al. (Hrsg.), *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte:* Springer Berlin Heidelberg, S. 639-660. DOI: 10.1007/978-3-662-45854-9_29.

Leudet, J., Christophe, F., Mikkonen, T. & Mannisto, T. (2019): AILiveSim: An Extensible Virtual Environment for Training Autonomous Vehicles. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC):* IEEE, S. 479-488. DOI: 10.1109/COMPSAC.2019.00074.

Leveson, N. (2020): Are you sure your software will not kill anyone? *Communications of the ACM (Commun. ACM).* Association for Computing Machinery (ACM), 63 (2), S. 25–28. DOI: 10.1145/3376127.

Leveson, N. G. (2012): *Engineering a Safer World - Systems Thinking Applied to Safety.* The MIT Press: Engineering Systems. DOI: 78510.

Lighthill, M. J. & Witham, G. B. (1955): On kinematic waves: II. A theory of traffic flow on long crowded roads. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 229 (1178), S. 317–345. DOI: 10.1098/rspa.1955.0089.

Liu, H., Deng, H., Li, J., Yang, S., Dong, K. & Zhao, Y. (2024a): Calibration method for microscopic traffic simulation considering lane difference. *SIMULATION*. DOI: 10.1177/00375497241268740.

Liu, J., Zhang, R., Yan, W., Zhao, Q. & Guo, C. (2024b): Modeling and simulation of fire evacuation considering guiding factors: a case study of Shenyang subway interchange station. *SIMULATION*. DOI: 10.1177/00375497241236969.

Liu, S., Xing, B., Li, B. & Gu, M. (2014): Ship information system: overview and research trends. *International Journal of Naval Architecture and Ocean Engineering*, 6 (3), S. 670–684. DOI: 10.2478/IJNAOE-2013-0204.

Liu, Z., Chu, Y., Li, G., Hildre, H. P. & Zhang, H. (2024c): A co-simulation approach to onboard support of marine operation: a Palfinger crane path planning case. *SIMULATION*, 100 (9), S. 919–930. DOI: 10.1177/00375497241228623.

Lloyd's Register (2019): *ShipRight Design and Construction - Procedure for assignment of digital descriptive notes for autonomous and remote access ships* (15/03/2019). Lloyd's Register.

Loehle, C. (2011): Complexity and the problem of ill-posed questions in ecology. *Ecological Complexity*, 8 (1), S. 60–67. DOI: 10.1016/j.ecocom.2010.11.004.

Lou, G., Deng, Y., Zheng, X., Zhang, M. & Zhang, T. (2022): Testing of autonomous driving systems: where are we and where should we go? In Roychoudhury, A., Cadar, C. & Kim, M. (Hrsg.), *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering:* ACM, S. 31-43. DOI: 10.1145/3540250.3549111.

Lunze, J. (2017): *Ereignisdiskrete Systeme - Modellierung und Analyse dynamischer Systeme mit Automaten, Markovketten und Petrinetzen* (3., überarbeitete Auflage). De Gryuter Oldenbourg: De Gruyter Studium Oldenbourg. ISBN: 9783110484717

Ma, J., Che, X., Li, Y. & Lai, E. M.-K. (2021): Traffic Scenarios for Automated Vehicle Testing: A Review of Description Languages and Systems. *Machines*, 9 (12). DOI: 10.3390/machines9120342.

Maier, M. W. (1998): Architecting principles for systems-of-systems. *Systems Engineering*, 1 (4), S. 267–284. DOI: 10.1002/(SICI)1520-6858(1998)1:4<267::AID-SYS3>3.0.CO;2-D.

Majchrzak, T. A. (2012): *Improving Software Testing - Technical and Organizational Developments.* Springer: SpringerBriefs in Information Systems. DOI: 10.1007/978-3-642-27464-0.

Majumdar, R., Mathur, A., Pirron, M., Stegner, L. & Zufferey, D. (2021): Paracosm: A Test Framework for Autonomous Driving Simulations. In Guerra, E. & Stoelinga, M. (Hrsg.), *Fundamental Approaches to Software Engineering:* Springer International Publishing, S. 172-195. ISBN: 978-3-030-71500-7.

Malinga, L. & Le Roux, W. H. (2009): HLA RTI performance evaluation. In *Proceedings of the 2009 SISO European Simulation Interoperability Workshop:* Society for Modeling & Simulation International, S. 39-44.

Mallozzi, P., Pelliccione, P., Knauss, A., Berger, C. & Mohammadiha, N. (2019): Autonomous Vehicles: State of the Art, Future Trends, and Challenges. In Dajsuren, Y. & van den Brand, M. (Hrsg.), *Automotive Systems and Software Engineering:* Springer International Publishing, S. 347-367. DOI: 10.1007/978-3-030-12157-0_16.

Mansfield, T., Caamaño, P., Godfrey, S. B., Carrera, A., Tremori, A., Nandakumar, G., Moberly, K., Cronin, J. & Da Deppo, S. (2022): Building Trust in Autonomous Systems: Opportunities for Modelling and Simulation. In Mazal, J. et al. (Hrsg.), *Modelling and Simulation for Autonomous Systems:* Springer International Publishing, S. 424-439. DOI: 10.1007/978-3-030-98260-7_27.

Mansouri, M., Gorod, A., Wakeman, T. H. & Sauser, B. (2009): Maritime Transportation System-of-Systems management framework: a System-of-Systems Engineering approach. *International Journal of Ocean Systems Management*, 1 (2), S. 200. DOI: 10.1504/IJOSM.2009.030185.

Maropoulos, P. G. & Ceglarek, D. (2010): Design verification and validation in product lifecycle. *CIRP Annals*, 59 (2), S. 740–759. DOI: 10.1016/j.cirp.2010.05.005.

Martinez, A. L., Champagne, L. E. & LaCasse, P. M. (2024): Simulating autonomous drone behaviors in an anti-access area denial (A2AD) environment. *The Journal of Defense Modeling and Simulation: Applications, Methodology, Technology.* SAGE Publications. DOI: 10.1177/15485129241288236.

Massei, M., Tremori, A., Poggi, S. & Nicoletti, L. (2013): HLA-based real time distributed simulation of a marine port for training purposes. *International Journal of Simulation and Process Modelling (IJSPM)*, 8 (1), S. 42. DOI: 10.1504/IJSPM.2013.055206.

Math, R., Mahr, A., Moniri, M. M. & Müller, C. (2013): OpenDS: A new open-source driving simulator for research. *GMM-Fachbericht-AmE 2013*, 2.

Mattern, F. (1996): Modellbildung und Simulation. In Wilhelm, R. (Hrsg.), *Informatik: Grundlagen - Anwendungen - Perspektiven [Forum "Perspektiven der Informatik", Dagstuhl, November 1993]:* Verlag C. H. Beck, S. 56-64. ISBN: 3-406-40338-7.

Mellor, A. (2023): *Test-driven development with Java - Create higher-quality software by writing tests first with solid and hexagonal architecture.* Packt Publishing. ISBN: 9781803236230

Menzel, T., Bagschik, G. & Maurer, M. (2018): Scenarios for Development, Test and Validation of Automated Vehicles. In *2018 IEEE Intelligent Vehicles Symposium (IV):* IEEE. DOI: 10.1109/IVS.2018.8500406.

Merali, Z. (2010): Computational science: …Error. *Nature*, 467 (7317), S. 775–777. DOI: 10.1038/467775a.

Meyer, B. (2008): Seven Principles of Software Testing. *Computer*, 41 (8), S. 99–101. DOI: 10.1109/MC.2008.306.

Miller, J. (2020): *Automotive system safety - Critical Considerations for Engineering and Effective Management.* John Wiley & Sons: Wiley series in quality & reliability engineering. DOI: 10.1002/9781119579663.ch3.

Mittelstadt, B., Russell, C. & Wachter, S. (2019): Explaining Explanations in AI. In *Proceedings of the Conference on Fairness, Accountability, and Transparency:* ACM, S. 279-288. DOI: 10.1145/3287560.3287574.

Mohan, R. & Ramadurai, G. (2013): State-of-the art of macroscopic traffic flow modelling. *International Journal of Advances in Engineering Sciences and Applied Mathematics*, 5 (2-3), S. 158–176. DOI: 10.1007/s12572-013-0087-1.

Möller, B. & Antelius, F. (2010): Object-Oriented HLA: Does One Size Fit All? In *Spring Simulation Interoperability Workshop 2010 (2010 Spring SIW):* Curran Associates, Inc., S. 411-420. ISBN: 9781617381133.

Möller, B., Löfstrand, B. & Karlsson, M. (2007): An overview of the HLA evolved modular FOMs [online]. In *Spring Simulation Interoperability Workshop*.

Moreira, E., Leite, J. R., Branco, M. G. & Ursini, E. L. (2025): Modeling and simulation for delay estimation of a generic packet transferred in a satellite data network. *SIMULATION*. DOI: 10.1177/00375497251322572.

Morse, K. L. & Steinman, J. S. (1997): DATA DISTRIBUTION MANAGEMENT IN THE HLA: Multidimensional Regions and Physically Correct Filtering [online]. In *Proceedings of the 1997 Spring Simulation Interoperability Workshop:* Simulation Interoperability Standards Organization (SISO).

Munim, Z. H. & Haralambides, H. (2022): Advances in maritime autonomous surface ships (MASS) in merchant shipping. *Maritime Economics & Logistics*, 24 (2), S. 181–188. DOI: 10.1057/s41278-022-00232-y.

Munim, Z. H. (2019): Autonomous ships: a review, innovative applications and future maritime business models. *Supply Chain Forum: An International Journal*, 20 (4), S. 266–279. DOI: 10.1080/16258312.2019.1631714.

Myers, G. J. (1979): *The Art of Software Testing.* Wiley: Business data processing, a Wiley series. ISBN: 0471043281

Myers, G. J., Corey Sandler & Tom Badgett (2012): *The Art of Software Testing* (3rd ed.). John Wiley & Sons. DOI: 10.1002/9781119202486.

Myklebust, T. & Stålhane, T. (2021): *Functional Safety and Proof of Compliance.* Springer International Publishing. DOI: 10.1007/978-3-030-86152-0.

Nalic, D., Mihalj, T., Eichberger, A., Bäumler, M. & Lehmann, M. (2021): Scenario Based Testing of Automated Driving Systems: A Literature Survey. In *FISITA World Congress 2021 - Technical Programme:* FISITA. DOI: 10.46720/f2020-acm-096.

National Highway Traffic Safety Administration (NHTSA) (2017): *Automated Driving Systems 2.0 - A Vision for Safety* [online]. United States Department of Transportation (USDOT), (DOT HS 812 442). Verfügbar unter: https://transportation.gov/av/2.0 [letzter Zugriff: 31/01/23].

Neurohr, C., Westhofen, L., Henning, T., Graaff, T. de, Mohlmann, E. & Bode, E. (2020): Fundamental Considerations around Scenario-Based Testing for Automated Driving. In *2020 IEEE Intelligent Vehicles Symposium (IV):* IEEE, S. 121-127. DOI: 10.1109/IV47402.2020.9304823.

Ni, D. (2003): 2DSIM: A prototype of nanoscopic traffic simulation. In *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683):* IEEE, S. 47-52. DOI: 10.1109/IVS.2003.1212881.

Nidhra, S. (2012): Black Box and White Box Testing Techniques - A Literature Review. *International Journal of Embedded Systems and Applications*, 2 (2), S. 29–50. DOI: 10.5121/ijesa.2012.2204.

Nielsen, C. B., Larsen, P. G., Fitzgerald, J., Woodcock, J. & Peleska, J. (2015): Systems of Systems Engineering: Basic Concepts, Model-Based Techniques, and Research Directions. *ACM Computing Surveys*, 48 (2), S. 1–41. DOI: 10.1145/2794381.

Nobis, P. (2016): *Entwicklung und Anwendung eines Modells zur Analyse der Netzstabilität in Wohngebieten mit Elektrofahrzeugen, Hausspeichersystemen und PV-Anlagen* [online]. doctoral. Verfügbar unter: https://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20160607-1276473-1-4 [letzter Zugriff: 26. März 2024].

Ofenloch, A., Schwarz, J. S., Tolk, D., Brandt, T., Eilers, R., Ramirez, R., Raub, T. & Lehnhoff, S. (2022): MOSAIK 3.0: Combining Time-Stepped and Discrete Event Simulation. In *2022 Open Source Modelling and Simulation of Energy Systems (OSMSES):* IEEE, S. 1-5. DOI: 10.1109/OSMSES54027.2022.9769116.

OFFIS e. V. (2018): *Architecture and Technology Development Platform for Realtime Safe and Secure Systems (ACTRESS) - Ensuring the dependability of modern open and adaptive maritime systems* [online]. The Actress Research Collaboration. Verfügbar unter: https://emaritime.de/actress/ [letzter Zugriff: 12. Oktober 2023].

OMG, UML:2.5.1. Object Management Group (OMG) (2017)**:** *Unified Modeling Language (UML) - formal/2017-12-05.* Verfügbar unter: https://omg.org/spec/UML/ [letzter Zugriff: 14. April 2021].

One Sea (2022): *AUTONOMOUS SHIPS - TERMS OF REFERENCE FOR RULE DEVELOPMENT* [online]. One Sea Association. Verfügbar unter: https://one-sea.org/documents/ [letzter Zugriff: 27. September 2024].

Oppelt, M., Wolf, G. & Urbas, L. (2014): Capability-analysis of co-simulation approaches for process industries. In *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA):* IEEE. DOI: 10.1109/ETFA.2014.7005292.

Oppermann, R. et al. (2019): *Software-ergonomische Evaluation - Der Leitfaden EVADIS II* (2. neubearb. Aufl. Früher u.d.T.: Evaluation von Dialogsystemen. Der Software-ergonomische Leitfaden EVADIS. Reprint 2019). De Gruyter Brill: Mensch, Computer, Kommunikation - Grundwissen. 5/2. DOI: 10.1515/9783110871524.

Özkanlisoy, Ö. & Akkartal, E. (2022): The Effect of Suez Canal Blockage on Supply Chains. *Maritime Faculty Journal.* Dokuz Eylül University, 14 (1), S. 51–79. DOI: 10.18613/deudfd.933816.

Pan, K., Turner, S. J., Cai, W. & Li, Z. (2007): A Service Oriented HLA RTI on the Grid. In *IEEE International Conference on Web Services (ICWS 2007):* IEEE, S. 984-992. DOI: 10.1109/ICWS.2007.20.

Parasuraman, R., Sheridan, T. B. & Wickens, C. D. (2000): A model for types and levels of human interaction with automation. *IEEE transactions on systems, man, and cybernetics. Part A, Systems and humans : a publication of the IEEE Systems, Man, and Cybernetics Society*, 30 (3), S. 286–297. DOI: 10.1109/3468.844354.

Partsch, H. A. (2010): *Requirements-Engineering systematisch.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-05358-0.

Patzak, G. (1982): *Systemtechnik - Planung komplexer innovativer Systeme - Grundlagen, Methoden, Techniken.* Springer. ISBN: 978-3-540-11783-4

Pecheur, C. (2000): *Verification and Validation of Autonomy Software at NASA* [online]. National Aeronautics and Space Administration (NASA): Computer Programming and Software, (NASA/TM-2000-209602). Verfügbar unter: https://ntrs.nasa.gov/citations/20010000882 [letzter Zugriff: 28. Februar 2025].

Pedersen, T. A., Glomsrud, J. A., Ruud, E.-L., Simonsen, A., Sandrib, J. & Eriksen, B.-O. H. (2020): Towards simulation-based verification of autonomous navigation systems. *Safety Science*, 129, S. 104799. DOI: 10.1016/j.ssci.2020.104799.

Penmetsa, P., Sheinidashtegol, P., Musaev, A., Adanu, E. K. & Hudnall, M. (2021): Effects of the autonomous vehicle crashes on public perception of the technology. *IATSS Research*, 45 (4), S. 485–492. DOI: 10.1016/j.iatssr.2021.04.003.

Penn, J. & Lin, A. (2016): The Trick Simulation Toolkit: A NASA/Opensource Framework for Running Time Based Physics Models. In *AIAA Modeling and Simulation Technologies Conference:* American Institute of Aeronautics and Astronautics. DOI: 10.2514/6.2016-1187.

Petty, M. D. (2002): Comparing high level architecture data distribution management specifications 1.3 and 1516. *Simulation Practice and Theory*, 9 (3-5), S. 95–119. DOI: 10.1016/S0928-4869(01)00051-9.

Pfeffer, R. & Leichsenring, T. (2016): Continuous Development of Highly Automated Driving Functions with Vehicle-in-the-Loop Using the Example of Euro NCAP Scenarios. In Gühmann, C., Riese, J. & Rüden, K. von (Hrsg.), *Simulation and Testing for Vehicle Technology:* Springer International Publishing, S. 33-42. DOI: 10.1007/978-3-319-32345-9_4.

Pidd, M. (1996): Five simple principle of modelling. In Charnes, J.M. et al. (Hrsg.), *Proceedings of the 28th Conference on Winter Simulation:* IEEE Computer Society, S. 721-728. DOI: 10.1145/256562.256794.

Pilarski, S., Sidhu, A. & Varró, D. (2025): Combining simulation and reinforcement learning to reduce food waste in food retail. *SIMULATION*, 101 (3), S. 267–285. DOI: 10.1177/00375497241299054.

Pohl, K. & Rupp, C. (2021): *Basiswissen Requirements Engineering - Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level* (5., überarbeitete und aktualisierte Auflage). dpunkt.verlag. ISBN: 9783864908149

Ponn, T., Breitfuß, M., Yu, X. & Diermeyer, F. (2020): Identification of Challenging Highway-Scenarios for the Safety Validation of Automated Vehicles Based on Real Driving Data. In *2020 Fifteenth International Conference on Ecological Vehicles and Renewable Energies (EVER),* S. 1-10. DOI: 10.1109/EVER48776.2020.9242539.

Porres, I., Azimi, S. & Lilius, J. (2020a): Scenario-based Testing of a Ship Collision Avoidance System. In Wimmer, M. & Skavhaug, A. (Hrsg.), *Proceedings, 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA 2020 : 26-28 August 2020, Kranj, Slovenia:* IEEE Computer Society, Conference Publishing Services, S. 545-552. DOI: 10.1109/SEAA51224.2020.00090.

Porres, I., Azimi, S., Lafond, S., Lilius, J., Salokannel, J. & Salokorpi, M. (2020b): On the Verification and Validation of AI Navigation Algorithms. In *Global Oceans 2020: Singapore – U.S. Gulf Coast:* IEEE, S. 1-8. DOI: 10.1109/IEEECONF38699.2020.9389133.

Pütz, A., Zlocki, A., Bock, J. & Eckstein, L. (2017): System validation of highly automated vehicles with a database of relevant traffic scenarios [online]. In *Proceedings of the 12th ITS European Congress*.

Raghupathi, W., Raghupathi, V. & Ren, J. (2022): Reproducibility in Computing Research: An Empirical Study. *IEEE Access*, 10, S. 29207–29223. DOI: 10.1109/ACCESS.2022.3158675.

Ramesh, V., Glass, R. L. & Vessey, I. (2004): Research in computer science: an empirical study. *Journal of Systems and Software*, 70 (1-2), S. 165–176. DOI: 10.1016/S0164-1212(03)00015-3.

Ramos, J. J. & Piera, M. A. (1999): Needs of object-oriented languages for physics knowledge representation in the simulation field. In Mitchell, R.J. (Hrsg.), *Technology of object-oriented languages and systems. TOOLS 29: Proceedings: June 7-10, 1999, Nancy, France:* IEEE Computer Society, S. 162-171. DOI: 10.1109/TOOLS.1999.779009.

Raza, M., Prokopova, H., Huseynzade, S., Azimi, S. & Lafond, S. (2022): Towards Integrated Digital-Twins: An Application Framework for Autonomous Maritime Surface Vessel Development. *Journal of Marine Science and Engineering*, 10 (10), S. 1469. DOI: 10.3390/jmse10101469.

Rehrl, K., Piribauer, T., Schwieger, K., Sedlacek, N. & Weissensteiner, P. (2020): Towards a uniform process model for deploying and operating autonomous shuttles on public roads. In *Proccedings 14th ITS European Congress*. DOI: 10.5281/ZENODO.4322876.

Reich, K. (2010): *Systemisch-konstruktivistische Pädagogik - Einführung in die Grundlagen einer interaktionistisch-konstruktivistischen Pädagogik.* Julius Beltz GmbH & Co. KG. ISBN: 3-407-25535-7

Reiher, D. & Hahn, A. (2021a): Review on the Current State of Scenario- and Simulation-Based V&V in Application for Maritime Traffic Systems. In *OCEANS 2021: San Diego – Porto:* IEEE. DOI: 10.23919/OCEANS44145.2021.9705781.

Reiher, D. & Hahn, A. (2021b): Towards a Model-Based Multi-Layered Approach to Describe Traffic Scenarios on a Technical Level. *Journal of Marine Science and Engineering (JMSE)*, 9 (6). DOI: 10.3390/jmse9060673.

Reite, K.-J., Føre, M., Aarsæther, K. G., Jensen, J., Rundtop, P., Kyllingstad, L. T., Endresen, P. C., Kristiansen, D., Johansen, V. & Fredheim, A. (2014): FHSIM — Time Domain Simulation of Marine Systems. In *Volume 8A: Ocean Engineering:* American Society of Mechanical Engineers. DOI: 10.1115/OMAE2014-23165.

Review of Maritime Transport 2019 **(2020).** United Nations Publications. ISBN: 978-92-1-112958-8

Riedmaier, S., Ponn, T., Ludwig, D., Schick, B. & Diermeyer, F. (2020): Survey on Scenario-Based Safety Assessment of Automated Vehicles. *IEEE Access*, 8, S. 87456–87477. DOI: 10.1109/ACCESS.2020.2993730.

Rodrigue, J.-P. (2013): *The Geography of Transport Systems* (3. Aufl.). Routledge. DOI: 10.4324/9781003343196.

Rødseth, Ø. J. & Nordahl, H. (2018): *Definition of autonomy levels for merchant ships, Report from NFAS, Norwegian Forum for Autonomous Ships, 2017-08-04.*

Rødseth, Ø. J. & Vagia, M. (2020): A taxonomy for autonomy in industrial autonomous mobile robots including autonomous merchant ships. *IOP Conference Series: Materials Science and Engineering*, 929 (1), S. 12003. DOI: 10.1088/1757-899X/929/1/012003.

Rødseth, Ø. J. & Wennersberg, L. A. (2023): A Criticism of Proposed Levels of Autonomy for MASS. In Brito, M.P. et al. (Hrsg.), *Proceeding of the 33rd European Safety and Reliability Conference:* Research Publishing Services, S. 2854-2860. DOI: 10.3850/978-981-18-8071-1_P090-cd.

Rødseth, Ø. J., Wennersberg, L. A. & Nordahl, H. (2022): Levels of autonomy for ships. *Journal of Physics: Conference Series*, 2311 (1), S. 12018. DOI: 10.1088/1742-6596/2311/1/012018.

Rong, G., Shin, B. H., Tabatabaee, H., Lu, Q., Lemke, S., Mozeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T. H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D. & Kim, S. (2020): LGSVL Simulator: A High Fidelity Simulator for Autonomous Driving. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC):* IEEE, S. 1-6. DOI: 10.1109/ITSC45102.2020.9294422.

Ropohl, G. (2009): *Allgemeine Technologie - Eine Systemtheorie der Technik* (3., überarbeitete Auflage). KIT Scientific Publishing; Universitätsverlag Karlsruhe: KIT Scientific Publishing. ISBN: 978-3-86644-374-7

Rosa, G., Scalabrino, S., Bavota, G. & Oliveto, R. (2023): What Quality Aspects Influence the Adoption of Docker Images? *ACM Transactions on Software Engineering and Methodology*, 32 (6), S. 1–30. DOI: 10.1145/3603111.

Ross, H.-L. (2016): *Functional Safety for Road Vehicles.* Springer International Publishing. DOI: 10.1007/978-3-319-33361-8.

Rowley, J. & Slack, F. (2004): Conducting a literature review. *Management Research News*, 27 (6), S. 31–39. DOI: 10.1108/01409170410784185.

Ruparelia, N. B. (2010): Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, 35 (3), S. 8–13. DOI: 10.1145/1764810.1764814.

Russell, S. J. & Norvig, P. (2016): *Artificial intelligence - A modern approach* (Third edition, global edition). Pearson Education Limited: Prentice Hall series in artificial intelligence. ISBN: 1292153962

Rüssmeier, N., Lamm, A. & Hahn, A. (2019): A generic testbed for simulation and physical-based testing of maritime cyber-physical System-of-Systems. *Journal of Physics: Conference Series*, 1357, S. 12025. DOI: 10.1088/1742-6596/1357/1/012025.

Ryan, V., Seligman, S. & Lee, R. (1999). *RFC 2713* [online] - *Schema for Representing Java(tm) Objects in an LDAP Directory.* Verfügbar unter: https://rfc-editor.org/rfc/rfc2713.html [letzter Zugriff: 11-09-23].

Sadjina, S., Kyllingstad, L. T., Rindarøy, M., Skjong, S., Æsøy, V. & Pedersen, E. (2019): Distributed Co-simulation of Maritime Systems and Operations. *Journal of Offshore Mechanics and Arctic Engineering*, 141 (1). DOI: 10.1115/1.4040473.

SAE, J3016:APR2021. SAE International (2021)**:** *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.* Verfügbar unter: https://sae.org/standards/content/j3016_202104/ [letzter Zugriff: 9. Dezember 2020].

SAE, J3016:JAN2014. SAE International (2014)**:** *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.* Verfügbar unter: https://sae.org/standards/content/j3016_201806/ [letzter Zugriff: 9. Dezember 2020].

Salem, N. F., Kirschbaum, T., Nolte, M., Lalitsch-Schneider, C., Graubohm, R., Reich, J. & Maurer, M. (2024): Risk Management Core - Towards an Explicit Representation of Risk in Automated Driving. *IEEE Access*, 12, S. 33200–33217. DOI: 10.1109/ACCESS.2024.3372860.

Santoro, A. & Fujimoto, R. M. (2008): Offloading Data Distribution Management to Network Processors in HLA-Based Distributed Simulations. *IEEE Transactions on Parallel and Distributed Systems*, 19 (3), S. 289–298. DOI: 10.1109/TPDS.2007.70715.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D. & Batra, D. (2019): Habitat: A Platform for Embodied AI Research. In *Proceedings of the International Conference on Computer Vision (ICCV) 2019:* IEEE, S. 9338-9346. DOI: 10.1109/ICCV.2019.00943.

Schaefer, K. E., Chen, J. Y., Szalma, J. L. & Hancock, P. A. (2016): A Meta-Analysis of Factors Influencing the Development of Trust in Automation: Implications for Understanding Autonomy in Future Systems. *Human factors*, 58 (3), S. 377–400. DOI: 10.1177/0018720816634228.

Scholtes, M., Westhofen, L., Turner, L. R., Lotto, K., Schuldes, M., Weber, H., Wagener, N., Neurohr, C., Bollmann, M. H., Kortke, F., Hiller, J., Hoss, M., Bock, J. & Eckstein, L. (2021): 6-Layer Model for a Structured De-

scription and Categorization of Urban Traffic and Environment. *IEEE Access*, 9, S. 59131–59147. DOI: 10.1109/ACCESS.2021.3072739.

Schuldt, F. (2017): *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen.* Dissertation. DOI: 10.24355/dbbs.084-201704241210.

Schuldt, F., Saust, F., Lichte, B., Maurer, M. & Scholz, S. (2013): Effiziente systematische Testgenerierung für Fahrerassistenzsysteme in virtuellen Umgebungen. *Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel (AAET).* ITS Mobility Cluster Niedersachsen, 4, S. 114–134.

Schulze, T., Strassburger, S. & Klein, U. (1999): Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications. *SIMULATION*, 73 (5), S. 296–303. DOI: 10.1177/003754979907300506.

Schulz-Schaeffer, I. (1998): Akteure, Aktanten und Agenten: konstruktive und rekonstruktive Bemühungen um die Handlungsfähigkeit von Technik. In Malsch, T. (Hrsg.), *Sozionik : soziologische Ansichten über künstliche Sozialität:* Ed. Sigma, S. 128-167. ISBN: 3-89404-453-5.

Schweigert, S. (2017): *Simulative Überprüfung von Sensordatenverarbeitungssystemen.* Oldenburg.

Schweigert, S., Gollücke, V., Hahn, A. & Bolles, A. (2014): Haggis: A modelling and simulation platform for e-Maritime technology assessment. In *INT-NAM 2014, 2nd International Symposium on Naval Architecture and Maritime:* Yıldız Technical Univ, S. 733-742. ISBN: 978-605-4123-32-2.

Serena, L., Marzolla, M., D'Angelo, G. & Ferretti, S. (2023): A review of multilevel modeling and simulation for human mobility and behavior. *Simulation Modelling Practice and Theory*, 127, S. 102780. DOI: 10.1016/j.simpat.2023.102780.

Shah, S., Dey, D., Lovett, C. & Kapoor, A. (2018): AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In Hutter, M. & Siegwart, R. (Hrsg.), *Field and Service Robotics:* Springer International Publishing, S. 621-635. DOI: 10.1007/978-3-319-67361-5_40.

Simon, H. A. (1996): *The Science Of The Artificial.* MIT Press. ISBN: 9780262190510

Singh, M., Fuenmayor, E., Hinchy, E., Qiao, Y., Murray, N. & Devine, D. (2021): Digital Twin: Origin to Future. *Applied System Innovation*, 4 (2), S. 36. DOI: 10.3390/asi4020036.

Singh, S. & Saini, B. S. (2021): Autonomous cars: Recent developments, challenges, and possible solutions. *IOP Conference Series: Materials Science and Engineering*, 1022 (1), S. 12028. DOI: 10.1088/1757-899X/1022/1/012028.

SISO, 007:2008. Simulation Interoperability Standards Organization (SISO) (2015)**:** *Standard for Military Scenario Definition Language.*

Sivori, H. & Brunton, L. (2023): *OUT OF THE BOX - Implementing autonomy and assuring artificial intelligence in the maritime industry* [online]. Thetius& Lloyd's Register. Verfügbar unter: https://lr.org/en/knowledge/research-reports/2023/ai-and-autonomy/ [letzter Zugriff: 3. August 2023].

Sloman, A. & Logan, B. (1999): Building cognitively rich agents using the SIM_Agent toolkit. *Communications of the ACM (Commun. ACM).* Association for Computing Machinery (ACM), 42, 71-ff. DOI: 10.1145/295685.295704.

Sloman, A. & Poli, R. (1996): SIM_AGENT: A toolkit for exploring agent designs. In Carbonell, J.G. et al. (Hrsg.), *Intelligent Agents II Agent Theories, Architectures, and Languages:* Springer Berlin Heidelberg, S. 392-407. DOI: 10.1007/3540608052_80.

Smogeli, Ø. R., Ludvigsen, K. B., Jamt, L., Vik, B., Nordahl, H., Kyllingstad, L. T., Yum, K. K. & Zhang, H. (2020): Open Simulation Platform - An Open-Source Project for Maritime System Co-Simulation. In Volker Bertram (Hrsg.), (Hrsg.), *19th International Conference on Computer and IT Applications in the Maritime Industries. COMPIT'20:* Technische Universität Hamburg-Harburg, S. 239-253. ISBN: 978-3-89220-717-7.

Sobieraj, M. & Kotyński, D. (2024): Docker Performance Evaluation across Operating Systems. *Applied Sciences*, 14 (15), S. 6672. DOI: 10.3390/app14156672.

Sokolowski, J. A. & Banks, C. M. (2009): *Principles of modeling and simulation - A multidisciplinary approach.* John Wiley. ISBN: 9780470289433

Spillner, A. & Linz, T. (2012): *Basiswissen Softwaretest - Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard* (5. Aufl.). dpunkt.verlag: ISQL-Reihe. ISBN: 9783864900242

Stefan Riedmaier, Jonas Nesensohn, Christian Gutenkunst, Bernhard Schick, Tobias Düser & Houssem Abdellatif (2018): Validation of X-in-the-Loop Approaches for Virtual Homologation of Automated Driving Functions [online]. In *11. Grazer Symposium VIRTUAL VEHICLE (GSVF) Graz, 15./16.05.2018*.

Steidel, M. & Hahn, A. (2019): MTCAS – An Assistance System for Collision Avoidance at Sea. In Bertram, V. (Hrsg.), *COMPIT'19. 18th International Conference on Computer and IT Applications in the Maritime Industries:* Technische Universität Hamburg-Harburg, S. 261-273. ISBN: 978-3-89220-709-2.

Steimle, M., Menzel, T. & Maurer, M. (2021): Toward a Consistent Taxonomy for Scenario-Based Development and Test Approaches for Automated Vehicles: A Proposal for a Structuring Framework, a Basic Vocabulary, and Its Application. *IEEE Access*, 9. DOI: 10.1109/ACCESS.2021.3123504.

Steinbrink, C., van der Meer, A. A., Cvetkovic, M., Babazadeh, D., Rohjans, S., Palensky, P. & Lehnhoff, S. (2018): Smart grid co-simulation with MOSAIK and HLA: a comparison study. *Computer Science - Research and Development*, 33 (1-2), S. 135–143. DOI: 10.1007/s00450-017-0379-y.

Sykes, P. (2010): Traffic Simulation with Paramics. In Barceló, J. (Hrsg.), *Fundamentals of Traffic Simulation:* Springer New York, S. 131-171. DOI: 10.1007/978-1-4419-6142-6_4.

Takács, Á., Drexler, D. A., Galambos, P., Rudas, I. J. & Haidegger, T. (2018): Assessment and Standardization of Autonomous Vehicles. In Szakál, A. (Hrsg.), *INES 2018. IEEE 22nd International Conference on Intelligent Engineering Systems:* IEEE, S. 185-192. DOI: 10.1109/INES.2018.8523899.

Tan, G. & Xu, L. (2001): An Agent-based Data Filtering Mechanism for High Level Architecture. *SIMULATION*, 76 (6), S. 329–344. DOI: 10.1177/003754970107600602.

Tan, G., Xu, L., Moradi, F. & Zhang, Y. (2000): An agent-based DDM filtering mechanism. In *Proceedings 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (Cat. No.PR00728):* IEEE Comput. Soc, S. 374-381. DOI: 10.1109/MASCOT.2000.876561.

Tang, J., Leu, G. & Abbass, H. A. (2019): Discrete Time Simulation. In Tang, J., Leu, G. & Abbass, H.A. (Hrsg.), *Simulation and Computational Red Teaming for Problem Solving:* Wiley, S. 143-155. DOI: 10.1002/9781119527183.ch8.

Teo, A., 2017: *The Maritime Journey Towards Autonomy - A Model-Based System Engineering Approach* [online]. Center of Excellence on Systems Architecture, Management, Economy and Strategy (CESAMES), 13. Dezember 2017: CESAM Community. Verfügbar unter: https://cesam.community/wp-content/uploads/2021/06/csdm_2017_The_Maritime_Journey_Towards_Autonomy_.pdf [letzter Zugriff: 27. September 2024].

THOMAS ROTH, MARTIN BURNS & TIM POKORNY **(Hrsg.) (2018):** *Extending Portico HLA to Federations of Federations with Transport Layer Security* [online]. 2018 Fall Simulation Innovation Workshop, Orlando, FL, US. Verfügbar unter: https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=926329

Thomke, S. & Fujimoto, T. (2000): The Effect of "Front-Loading" Problem-Solving on Product Development Performance. *Journal of Product Innovation Management*, 17 (2), S. 128–142. DOI: 10.1016/S0737-6782(99)00031-4.

Tibba, G., Malz, C., Stoermer, C., Nagarajan, N., Zhang, L. & Chakraborty, S. (2016): Testing automotive embedded systems under X-in-the-loop setups. In Liu, F. (Hrsg.), *Proceedings of the 35th International Conference on Computer-Aided Design:* ACM, S. 1-8. DOI: 10.1145/2966986.2980076.

Tolk (Hrsg.) (2001): *HLA-OMT versus Traditional Data and Object Modeling - Updated and Extended Version for the 6. ICCRTS* [online]. International Command and Control Institute (IC2I). Verfügbar unter: http://dodccrp.org/events/6th_ICCRTS/Tracks/T3.htm [letzter Zugriff: 11. August 2023].

Tolk (Hrsg.) (2012a): *Engineering principles of combat modeling and distributed simulation.* John Wiley & Sons, Inc. DOI: 10.1002/9781118180310.

Tolk, A. (2002): Avoiding another Green Elephant: A Proposal for the Next Generation HLA based on the Model Driven Architecture. In *2002 Fall Simulation Interopberability Workshop:* Simulation Interoperability Standards Organization (SISO).

Tolk, A. (2012b): Challenges of Combat Modeling and Distributed Simulation. In Tolk, A. (Hrsg.), *Engineering principles of combat modeling and distributed simulation:* John Wiley & Sons, Inc., S. 1-22. ISBN: 9780470874295.

Torben, T., Smogeli, O., Utne, I. & Sørensen, A. (2022): On Formal Methods for Design and Verification of Maritime Autonomous Surface Ships. In *World Maritime Technology Conference (WMTC):* Society of Naval Architecture and Marine Engineering (SNAME).

Treiber, M. & Kesting, A. (2010): *Verkehrsdynamik und -simulation.* Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-05228-6.

Tseng, S.-H. & Aldi Winata, O. (2025): Applying simulation technique to enhance the steel manufacturing process considering transportations. *SIMULATION*. DOI: 10.1177/00375497251326918.

Uhrmacher, A. M., Brailsford, S., Liu, J., Rabe, M. & Tolk, A. (2016): Panel — Reproducible research in discrete event simulation — A must or rather a maybe? In *2016 Winter Simulation Conference (WSC):* IEEE, S. 1301-1315. DOI: 10.1109/WSC.2016.7822185.

Ulbrich, S., Menzel, T., Reschka, A., Schuldt, F. & Maurer, M. (2015): Definition der Begriffe Szene, Situation und Szenario für das automatisierte Fahren. In *10. Workshop Fahrerassistenzsysteme. FAS 2015:* Uni-DAS e.V, S. 105-118. ISBN: 978-3-00-050746-5.

UNFCCC (2015): Paris Agreement [online]. In *United Nations Treaties*.

Uzuka, T. (2023): System-of-Systems in Railway. In Kaihara, T. et al. (Hrsg.), *Innovative Systems Approach for Facilitating Smarter World:* Springer Nature Singapore, S. 199-209. DOI: 10.1007/978-981-19-7776-3_16.

Vagia, M., Transeth, A. A. & Fjerdingen, S. A. (2016): A literature review on the levels of automation during the years. What are the different taxonomies that have been proposed? *Applied ergonomics*, 53 Pt A, S. 190–202. DOI: 10.1016/j.apergo.2015.09.013.

Valmari, A. (1998): The state explosion problem. In Goos, G. et al. (Hrsg.), *Lectures on Petri Nets I: Basic Models:* Springer Berlin Heidelberg, S. 429-528. DOI: 10.1007/3-540-65306-6_21.

Vangheluwe, H. & Lara, J. de (2002): Meta-models Are Models Too. In *Proceedings of the Winter Simulation Conference:* IEEE, S. 597-605. DOI: 10.1109/WSC.2002.1172936.

Varga, B., Tettamanti, T. & Szalay, Z. (2021): System Architecture for Scenario-In-The-Loop Automotive Testing. *Transport and Telecommunication Journal*, 22 (2), S. 141–151. DOI: 10.2478/ttj-2021-0011.

VDI, 2206. Verein Deutscher Ingenieure e. V. (VDI)& Verband der Elektrotechnik Elektronik Informationstechnik (VDE) (2021)**:** *Entwicklung mechatronischer und cyber-physischer Systeme.* Verfügbar unter: https://vdi.de/richtlinien/details/vdivde-2206-entwicklung-mechatronischer-und-cyber-physischer-systeme [letzter Zugriff: 31. Januar 2025].

VDI, 3363. Verein Deutscher Ingenieure e. V. (VDI) (2014)**:** *Simulation von Logistik-, Materialfluss und Produktionssystemen - Grundlagen.* Verfügbar unter: https://vdi.de/richtlinien/details/vdi-3633-blatt-1-simulation-von-logistik-materialfluss-und-produktionssystemen-grundlagen [letzter Zugriff: 31. Januar 2025].

Verma, S. B., Pandey, B. & Kumar Gupta, B. (2022): Containerization and its Architectures: A Study. *Advances in Distributed Computing and Artificial Intelligence Journal (ADCAIJ).* Salamanca University Press, 11 (4), S. 395–409. DOI: 10.14201/adcaij.28351.

vom Brocke, J., Simons, A., Niehaves, B., Riemer, K., Plattfaut, R. & Cleven, A. (2009): Reconstructing the Giant: On the Importance of Rigour in Documenting the Literature Search Process [online]. In *17th European Conference on Information Systems (ECIS 2009).* 17. Auflage.

Wachenfeld, W. & Winner, H. (2015a): Der Sicherheitsnachweis für autonome Fahrzeuge. In Hilgendorf, E., Hötitzsch, S. & Lutz, L.S. (Hrsg.), *Rechtliche Aspekte automatisierter Fahrzeuge:* Nomos, S. 53-60. DOI: 10.5771/9783845261638-53.

Wachenfeld, W. & Winner, H. (2015b): Die Freigabe des autonomen Fahrens. In Maurer, M. et al. (Hrsg.), *Autonomes Fahren: Technische, rechtliche und gesellschaftliche Aspekte:* Springer Berlin Heidelberg, S. 439-464. ISBN: 978-3-662-45854-9.

Wachenfeld, W. & Winner, H. (2016): The Release of Autonomous Vehicles. In Maurer, M. et al. (Hrsg.), *Autonomous Driving:* Springer Berlin Heidelberg, S. 425-449. DOI: 10.1007/978-3-662-48847-8_21.

Walker, T. R., Adebambo, O., Del Aguila Feijoo, M. C., Elhaimer, E., Hossain, T., Edwards, S. J., Morrison, C. E., Romo, J., Sharma, N., Taylor, S. & Zomorodi, S. (2019): Environmental Effects of Marine Transportation. In *World Seas: An Environmental Evaluation:* Elsevier, S. 505-530. DOI: 10.1016/B978-0-12-805052-1.00030-9.

Wallace, D. R. & Fujii, R. U. (1989): Software verification and validation: an overview. *IEEE Software*, 6 (3), S. 10–17. DOI: 10.1109/52.28119.

Wang, Y., Gong, J., Wang, B., Jia, P. & Kyo, T. (2022): Off-road Testing Scenario Design and Library Generation for Intelligent Vehicles. *Green Energy and Intelligent Transportation.* Elsevier. DOI: 10.1016/j.geits.2022.100013.

Watada, J., Roy, A., Kadikar, R., Pham, H. & Xu, B. (2019): Emerging Trends, Techniques and Open Issues of Containerization: A Review. *IEEE Access.* Institute of Electrical and Electronics Engineers (IEEE), 7, S. 152443–152472. DOI: 10.1109/ACCESS.2019.2945930.

Weber, H., Bock, J., Klimke, J., Roesener, C., Hiller, J., Krajewski, R., Zlocki, A. & Eckstein, L. (2019): A framework for definition of logical scenarios for safety assurance of automated driving. *Traffic injury prevention*, 20, S65-S70. DOI: 10.1080/15389588.2019.1630827.

Weber, N., Frerichs, D. & Eberle, U. (2020): A simulation-based, statistical approach for the derivation of concrete scenarios for the release of highly automated driving functions. In *GMM-Fachbericht 95. Automotive meets Electronics Beiträge:* VDE Verlag, S. 116-121. DOI: 10.13140/RG.2.2.15306.31683/1.

Webster, J. & Watson, R. T. (2002): Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly.* Management Information Systems Research Center, University of Minnesota, 26 (2), S. 8–13. Verfügbar unter: http://jstor.org/stable/4132319 [letzter Zugriff: 27. November 2020].

Wennersberg, L. A., Nordahl, H., Rødseth, Ø. J., Fjørtoft, K. & Holte, E. A. (2020): A framework for description of autonomous ship systems and operations. *IOP Conference Series: Materials Science and Engineering.* IOP Publishing, 929 (1), S. 12004. DOI: 10.1088/1757-899X/929/1/012004.

Westhofen, L., Neurohr, C., Koopmann, T., Butz, M., Schütt, B., Utesch, F., Neurohr, B., Gutenkunst, C. & Böde, E. (2023): Criticality Metrics for Automated Driving: A Review and Suitability Analysis of the State of the Art. *Archives of Computational Methods in Engineering*, 30 (1), S. 1–35. DOI: 10.1007/s11831-022-09788-7.

Wigan, M. R. (1970): Applications of simulation to traffic problems. *SIMULATION*, 14 (3), S. 135–136. DOI: 10.1177/003754977001400309.

Wilde, T. & Hess, T. (2006): *Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung* [online], (2/2006). Verfügbar unter: https://hdl.handle.net/10419/60077

Witte, F. (2016): *Testmanagement und Softwaretest.* Springer Fachmedien Wiesbaden. DOI: 10.1007/978-3-658-09964-0.

Wozniak, M. (2024): From dawn to dusk: daily fluctuations in pedestrian traffic in the city center. *SIMULATION*, 100 (3), S. 245–263. DOI: 10.1177/00375497231212543.

Wright, R. G. (2020): *Unmanned and autonomous ships - An overview of MASS.* Routledge Taylor & Francis Group. ISBN: 1138324884

Wu, H., Lyu, D., Zhang, Y., Hou, G., Watanabe, M., Wang, J. & Kong, W. (2022): A verification framework for behavioral safety of self-driving cars. *IET Intelligent Transport Systems*, 16 (5), S. 630–647. DOI: 10.1049/itr2.12162.

Wüllner, T., Feuerstack, S. & Hahn, A. (2019): Clustering Environmental Conditions of Historical Accident Data to Efficiently Generate Testing Sceneries for Maritime Systems. In Papadopoulos, Y. et al. (Hrsg.), *Model-Based Safety and Assessment:* Springer International Publishing, S. 349-362. ISBN: 978-3-030-32872-6.

Wurst, J., Balasubramanian, L., Botsch, M. & Utschick, W. (2022): Expert-LaSTS: Expert-Knowledge Guided Latent Space for Traffic Scenarios. In *2022 IEEE Intelligent Vehicles Symposium (IV):* IEEE, S. 484-491. DOI: 10.1109/IV51971.2022.9827187.

Wymann, B. et al. (2015). *TORCS: The open racing car simulator* [online]. Verfügbar unter: https://cse.chalmers.se/~chrdimi/papers/torcs.pdf [letzter Zugriff: 24-14-05].

Wymann, B. et al. (2016). TORCS [Software]. Version 1.3.7. Verfügbar unter: https://sourceforge.net/projects/torcs/ [letzter Zugriff: 14/04/25].

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J. & Savarese, S. (2018): Gibson Env: Real-World Perception for Embodied Agents. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition:* IEEE, S. 9068-9079. DOI: 10.1109/CVPR.2018.00945.

Xiao, F., Ligteringen, H., van Gulijk, C. & Ale, B. (2013): Nautical traffic simulation with multi-agent system for safety. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013):* IEEE, S. 1245-1252. DOI: 10.1109/ITSC.2013.6728402.

Xiao, Z., Fu, X., Zhang, L., Zhang, W., Agarwal, M., Siow, R. & Mong, G. (2019): MarineMAS: A multi-agent framework to aid design, modelling, and evaluation of autonomous shipping systems. *Journal of International Maritime Safety, Environmental Affairs, and Shipping.* Taylor & Francis Online, 2 (2), S. 43–57. DOI: 10.1080/25725084.2019.1569318.

Xu, L., Lin, S.-Y., Hlynka, A. W., Lu, H., Kamat, V. R., Menassa, C. C., El-Tawil, S., Prakash, A., Spence, S. M. & McCormick, J. (2021): Distributed Simulation Platforms and Data Passing Tools for Natural Hazards Engineering: Reviews, Limitations, and Recommendations. *International Journal of Disaster Risk Science*, 12 (5), S. 617–634. DOI: 10.1007/s13753-021-00361-7.

Yao, S., Zhang, J., Hu, Z., Wang, Y. & Zhou, X. (2018): Autonomous-driving vehicle test technology based on virtual reality. *The Journal of Engineering*, 2018 (16), S. 1768–1771. DOI: 10.1049/joe.2018.8303.

Yuri P., P. & Sizikov, V. S. (2005): *Well-posed, Ill-posed, and Intermediate Problems with Applications.* DE GRUYTER. DOI: 10.1515/9783110195309.

Zanella, T. V. (2020): THE ENVIRONMENTAL IMPACTS OF THE "MARITIME AUTONOMOUS SURFACE SHIPS" (MASS). *Veredas do Direito: Direito Ambiental e Desenvolvimento Sustentável*, 17 (39). DOI: 10.18623/rvd.v17i39.1803.

Zednik, C. (2021): Solving the Black Box Problem: A Normative Framework for Explainable Artificial Intelligence. *Philosophy & Technology*, 34 (2), S. 265–288. DOI: 10.1007/s13347-019-00382-7.

Zeigler, B. P., Nicolau de França, B. B., Graciano Neto, V. V., Hill, R. R., Champagne, L. E. & Ören, T. (2023): History of Simulation. In Ören, T., Zeigler, B.P. & Tolk, A. (Hrsg.), *Body of Knowledge for Modeling and Simulation:* Springer International Publishing, S. 413-434. DOI: 10.1007/978-3-031-11085-6_17.

Zhang, T., Liu, H., Wang, W. & Wang, X. (2024): Virtual Tools for Testing Autonomous Driving: A Survey and Benchmark of Simulators, Datasets, and Competitions. *Electronics*, 13 (17), S. 3486. DOI: 10.3390/electronics13173486.

ABBILDUNGEN

Abbildung 1: Begriffe mit Bezug zum Konzept der *Sicherheit* in ihren Menge-Teil-Beziehungen dargestellt. Das für diese Arbeit relevanteste Konzept der *Safety of the Intended Functionality (SOTIF)* ist durch Fettdruck hervorgehoben. 3

Abbildung 2: Einordnung der in dieser Arbeit betrachteten Szenariomodellierung in den szenariobasierten Entwicklungsprozess von automatisierten Fahrsystemen (grob angelehnt an den in [Hanke, 2020] vorgestellten Prozess). Die Teilprozesse, die Hauptgegenstand dieser Arbeit sind, sind fett umrandet. 10

Abbildung 3: Die relevantesten im Rahmen dieser Arbeit erzeugten Artefakte. Erkennbar sind die stringenten Zusammenhänge. Die Hypothese fungiert als Bindeglied zwischen den beiden Phasen dieser gestaltungsorientierten Forschungsarbeit und stellt die argumentative Grundlage zur Beantwortung der Forschungsfragen durch die Implementierung dar..... 20

Abbildung 4: Aufbau der vorliegenden Arbeit nach ingenieurswissenschaftlicher Methodik. Zusätzlich zu den aufeinander aufbauenden Kapitelinhalten sind Informationsflüsse und inhaltlichen Rückbezüge dargestellt. 23

Abbildung 5: Computergrafische 3D Illustration des Containerschiffes Yara Birkeland, welches zukünftig vollständig automatisiert und ohne Personal an Bord zwischen dem Herøya-Industriepark und den Häfen Brevik und Larvik verkehren soll. [Yara International ASA, 2022] 28

Abbildung 6: Taxonomie maritimer Fahrzeuge nach der Art und Weise, wie diese gesteuert und bedient werden. Die hellgrauen Rechtecke stellen bereits gebräuchliche Begriffe dar. Dunkelgraue Rechtecke repräsentieren die Arten maritimer Fahrzeuge, die im Fokus der Betrachtung dieser Arbeit stehen. Der abgeänderte Begriff *Maritime Automated Surface Ship* (MASS) umfasst einfachheitshalber sowohl die Kategorie *Automated Ship* als auch die Kategorie *Autonomous Ship*. 32

Abbildung 7: Abstraktes 4-Ebenen-Modell des menschlichen Informationsverarbeitungsprozesses nach [Parasuraman, Sheridan und Wickens, 2000] (Übers. d. Verf.) 33

Abbildung 8: Drei-Ebenen-Hierarchie für Fahraufgaben nach [Donges, 2015] angepasst auf die Konzepte und Eigenschaften maritimen Verkehrs (angelehnt an [Brinkmann, 2018] mit zusätzlicher strengerer Integration der Teilaufgaben eines üblichen GNC-Konzepts, wie es u. a. in [Fossen, 2011] beschrieben ist). Zu erkennen sind die Feedbackschleifen aus Wahrnehmung der Umgebung und Ausführung von Aktionen, welche sich auf die Umwelt auswirken. 35

Abbildung 9: Abgrenzung eines Systems zur (für das System relevanten) Umwelt 37

Abbildung 10: Komponenten eines Systems (angelehnt an [Dierßen, 2002]) 37

Abbildung 11: Minimalbeispiel einer Trajektorie in einem Zustandsraum: Der Zustand eines angenommenen Systems (z. B. ein Fahrzeug) definiert sich hier nur über dessen Position in seiner als planar angenommenen Umwelt. Die Position setzt sich entsprechend zusammen aus den X- und Y-Koordinaten in dieser Umwelt. Die Umwelt hat hier eine definierte Größe und die sich dadurch ergebenden Grenzen können von dem System nicht unter- oder überschritten werden. Der sich so ergebene Zustandsraum wird durch die X-Achse, die Y-Achse sowie die gestrichelten Linien dargestellt. 38

Abbildung 12: Ein offenes System als Blackbox 39

Abbildung 13: Stark vereinfachte Veranschaulichung eines Containerschiffes als System-of-Systems 40

Abbildung 14: Zusammensetzung und Beziehungen automatisierter Fahrzeuge, ihrer Funktionen und Bestandteile (angelehnt an [Steimle, Menzel und Maurer, 2021]) 41

Abbildung 15: Beziehungen der an der Konstruktion und dem Betrieb eines automatisierten Schiffs beteiligten Unternehmen untereinander und zu den (Teil-)Systemen (nach [Elhadeedy und Daily, 2024]) 42

Abbildung 16: Mögliche Bestandteile eines soziotechnischen System-of-Systems 43

Abbildung 17: Typische Ausprägung des V-Modells, wie es u. a. in der Systementwicklung zum Einsatz kommt. Jeder Entwurfsaktivität ist eine entsprechende Testaktivität zur Überprüfung der jeweiligen Ergebnisse fest zugeordnet. 44

Abbildung 18: Doppel-V-Modell zur Steuerung des Entwicklungsprozesses eines System-of-Systems (in Anlehnung an [Dahmann, 2015b]) 45

Abbildung 19: Durch Integration individueller V&V-Feedback-Schleifen in die einzelnen Entwurfs- und Umsetzungsphasen des Entwicklungsprozesses von automatisierten Fahrzeugsystemen angepasstes V-Modell (Idee und Abbildung basierend auf den in [Koopman und Wagner, 2016] und [Brinkmann, 2018] vorgeschlagenen Prozessen). Phasen außerhalb des Betrachtungsumfangs dieser Arbeit sind grau dargestellt. 50

Abbildung 20: Relevante Begriffe zur Beschreibung von Fehlern, ihren Ursachen und Auswirkungen softwarebasierter Systeme. (Angelehnt an [Majchrzak, 2012]; deutschsprachige Begriffe übernommen aus [Spillner und Linz, 2012];) 52

Abbildung 21: Klassifikation von Methoden zum Testen von Software(-basierten) Systemen und Komponenten. Die dargestellten Kategorien sind in der Realität nicht alle scharf voneinander abgegrenzt. So kann ein funktionaler Test ebenso nichtfunktionale Aspekte überprüfen. Auch im Grenzbereich zwischen Blackbox-Tests und Whitebox-Tests hat sich eine weitere Kategorie etabliert: der Greybox-Test. 53

Abbildung 22: Mögliche Zustände des Philosophenproblems mit (a) einem, (b) zwei und (c) drei Philosophen, die jeweils einen der vier Zustände DENKEN, HUNGER, LINKS, ESSEN annehmen können. Dabei bedeutet LINKS, dass das linke Essbesteck genommen wurde und ESSEN, dass sowohl das Essbesteck zur Linken als auch das zur Rechten gehalten wird. Zur Verbesse-

rung der Übersichtlichkeit sind die meisten rückführenden Zustandsübergänge nicht dargestellt. Aus gleichem Grund ist in (c) der Zustandsraum nach unten offen dargestellt. 55

Abbildung 23: Die Beziehungen zwischen Situationen, Szenen und Szenario. Das Fortschreiten der Zeit wird von der horizontalen Achse repräsentiert, wobei t_0 der Anfangszustand ist. Die Menge aller verfügbaren Informationen über alle Akteure und ihre Umgebung zu einem bestimmten Zeitpunkt *t* bildet den Inhalt einer Szene. Für jeden Akteur innerhalb dieser Szene existiert zum selben Zeitpunkt *t* eine Situation als Teilmenge der in der Szene enthaltenen Informationen. Ein Szenario ist die Kombination einer Startszene und Zielen, Werten sowie Verhaltensbeschreibungen der Akteure. 59

Abbildung 24: Struktur eines Szenarios und seiner Inhalte (Inhaltlich basierend auf [Ulbrich et al., 2015b]; Grafisch angelehnt an [Steimle, Menzel und Maurer, 2021]) 61

Abbildung 25: Zusammenhang der Begriffe (System-)Modell, (Modell-)Zustand, Simulationslauf und Zustandstrajektorie im Kontext einer zeitbasierten Simulation 66

Abbildung 26: Vereinfachtes Model des *Modelling & Simulation* Prozesses (angelehnt an [Ali et al., 2024]) 67

Abbildung 27: Modellierung von dynamischem Verhalten: (a) beschreibendes Verhaltensmodell, (b) erklärendes Verhaltensmodell (angelehnt an [Bossel, 2004]) 69

Abbildung 28: Ein mit der Zeit zunehmender Verlauf einer Zustandsgröße. Bei einer *zeitdiskreten Simulation* (links) (engl.: Discrete-Time Simulation, DTS) ist der Zustand nur zu diskreten Zeitpunkten definiert. Bei einem *zeitkontinuierlichen Modell* (engl.: Continous-Time Simulation, CTS) (rechts) ist der Zustand hingegen zu jedem beliebigen Zeitpunkt definiert, wodurch der Verlauf keine Lücken aufweist. 71

Abbildung 29: Beispielhafter Verlauf einer *zustandsdiskreten* Simulation (auch *ereignisdiskrete* Simulation genannt, engl.: Discrete-Event Simulation, DES). Die Markierungen an der x-Achse e_1 bis e_6 markieren den Zeitpunkt des Eintretens des jeweiligen Ereignisses. 72

Abbildung 30: Visualisierung der drei etablierten Granularitäten von Verkehrssimulationen: makroskopisch, mikroskopisch, submikroskopisch (v. l. n. r.) 74

Abbildung 31: Abstrakte Darstellung eines einfachen nicht näher konkretisierten Agenten als Teil eines Simulationslaufs (angelehnt an [Russell und Norvig, 2016]). Das dynamische Verhalten des Agenten wird durch das interne Modell bestimmt, welches die Rolle der von Russell und Norvig als *Agent Program* bezeichneten Implementierung einnimmt.... 77

Abbildung 32: Abstrakte Darstellung eines Agenten als Repräsentation eines System-of-Systems (vgl. Abbildung 31). 78

Abbildung 33: Ein Tupel von *Verkehrslagen* als Ergebnis eines zeitdiskreten (sub-)mikroskopischen Simulationslaufs. 79

Abbildung 34: Bidirektionale Einbindung einer zu testenden Komponente in einen Simulationslauf..... 79

Abbildung 35: Die Modelle und Artefakte eines szenariobasierten Simulationslaufs (vgl. Abbildung 2) 81

Abbildung 36: Beziehungen des Simulationsmodells einer monolithischen Simulationsanwendungen..... 82

Abbildung 37: Einordnung des vorgeschlagenen Begriffs einer Szenarioinstanz..... 84

Abbildung 38: Beziehungen zwischen den Modellen und ihrer Bestandteile, die für die Durchführung einer szenariobasierten Simulation notwendig sind. 85

Abbildung 39: Orthogonale Modellierungshierarchien (engl.: Modelling Spaces) einer klassischen szenariobasierten Verkehrssimulation mit getrennten, eigenständigen Szenario- und Simulationsmodellen..... 87

Abbildung 40: Anforderungsdiagramm zur Darstellung der direkten Beziehung jedes gestaltungsorientierten Ziels zu einer der formulierten Handlungsempfehlungen. 104

Abbildung 41: Anforderungsdiagramm zur Darstellung der hierarchischen Abhängigkeiten zwischen den zu Beginn aufgestellten Forschungsfragen und den Zielen, die zur Beantwortung dieser im praktischen Teil verfolgt werden..... 106

Abbildung 42: Anwendungsfalldiagramm zur Darstellung der Rollen und Tätigkeiten, die Teil eines V&V-integrierten Entwicklungsprozesses beim Einsatz von szenariobasierten Simulationsläufen sind, und der Beziehungen zwischen ihnen. Die dargestellten Tätigkeiten stellen eine Verfeinerung des zu Beginn dieser Arbeit in Abbildung 2 dargestellten allgemeinen szenariobasierten Entwicklungsprozess dar. 109

Abbildung 43: Angepasste Variante der Anforderungsschablone für funktionale Anforderungen ohne zusätzliche Bedingung nach [Pohl & Rupp, 2021]. Die Menge der Modalverben wurde gemäß [Koelsch, 2016] auf *muss* reduziert. Bei der Verwendung der Schablone sind Begriffe, die in ⟨WINKELKLAMMERN⟩ dargestellt sind, Platzhalter und entsprechend der zu formulierenden Anforderung zu ersetzen..... 110

Abbildung 44: Anforderungsdiagramm zur Darstellung der identifizierten lösungsorientierten Anforderungen, die an die zur Erreichung der gestaltungsorientierten Ziele entwickelten technischen Konzepte gestellt werden. 115

Abbildung 45: Individuelle Verkehrsteilnehmer als Agenten einer submikroskopischen Verkehrssimulation. Die modellierte Wirkstruktur eines Verkehrsteilnehmers (VT) kann unterschiedliche Ausprägungen bezüglich des Detailgrads annehmen. Systemmodelle (SM) können durch eine Menge von Teilsystemmodellen (TSM) verfeinert werden, welche wiederum durch eine Menge von Komponentenmodellen (KM) abgebildet werden können. 117

Abbildung 46: Das Konzept der Metamodellierung, wie es im Meta-Object-Facility (MOF)-Standard für modellgetriebene Entwicklung beschrieben wird, angereichert mit einem vereinfachten UML-basierten Minimalbeispiel. 119

Abbildung 47: Inhalte eines Szenarios, erweitert um geschachtelte Komponenten innerhalb eines Akteurs, zur Abbildung der internen Wirkstrukturen von modernen Fahrzeugen (vgl. Abbildung 24) 120

Abbildung 48: Einfluss der Ziele eines Tests auf den Umfang und den Detailgrad des optimalen Modells 121

Abbildung 49: Ontologische Modellhierarchie eingebettet in die Ebene der linguistischen Modellhierarchie. Durch die konzeptorientierte Trennung der Modellinhalte auf die Schichten O2, O1 und O0 wird die Wiederverwendung der tieferen Schichten ermöglicht. Auf der rechten Seite ist jeweils ein Beispiel-Element für jede Schicht abgebildet. 122

Abbildung 50: Mögliche inhaltliche Vererbungsstruktur innerhalb der Domänen-Ebene 122

Abbildung 51: Mögliche Aufteilung der Elemente des Domänenmodells auf die drei Teilpakete D**1** bis D**3** 122

Abbildung 52: Referenz- und Nutzungsbeziehungen der Modelle eines szenariobasierten Simulationslaufs. 123

Abbildung 53: Zusammengeführte Szenario- und Simulationsmodelle. Das Laufzeitmodell wird mit Hilfe der Szenarioinstanz aus den Modellelementen erzeugt und existiert nur flüchtig (transientes Simulationsmodell). 123

Abbildung 54: Transientes Simulationsmodell 124

Abbildung 55: Verantwortlichkeiten in Bezug auf die Teilmodelle der Modellierungshierarchie 125

Abbildung 56: Modularisierung der Modellierungshierarchie 125

Abbildung 57: Agentengerüst zur Kapselung der Kommunikations- und Steuerungskomplexität 126

Abbildung 58: Szenariobibliothek als Basis für die Spezifikation der Szenarioinstanz 127

Abbildung 59: Erweiterung der möglichen Szenarioinhalte um eine Repräsentation der möglichen Testobjekte 128

Abbildung 60: Netzwerkkommunikation zwischen der Simulationsanwendung und dem externen Testobjekt. 129

Abbildung 61: Anbindung eines externen Testobjektes in der Rolle eines Akteurs 131

Abbildung 62: Anbindung eines externen Testobjektes in der Rolle einer Komponente 131

Abbildung 63: Überblick über die entworfene Systemumgebung szenariobasierter Simulationsläufe zu Testzwecken softwarebasierter Systeme oder Komponenten dieser. 133

Abbildung 64: Aufbau einer Co-Simulation nach dem FMI-CS-Standard 137

Abbildung 65: Aufbau einer verteilten Simulation nach dem HLA-Standard 141

Abbildung 66: Metamodell – abstrakte Superklasse aller möglichen Inhalte einer Szenarioinstanz 144

Abbildung 67: Metamodell – Hierarchie physischer Charakteristiken 145

Abbildung 68: Metamodell – verhaltensbezogene Charakteristiken 146

Abbildung 69: Metamodell – Komponente 147

Abbildung 70: Metamodell – Basiselemente *ScenarioSpecificationItem*, *Element* und *Property* 148

Abbildung 71: Metamodell – *Unit* und *Prefix* als Repräsentation des Internationale Einheitensystems bilden die Grundlage für die Umrechnungen unterschiedliche physikalischer Größen derselben Dimension 148

Abbildung 72: Metamodell – *Position* als konkretes Beispiel zusammengesetzter Datentypen 148

Abbildung 73: Metamodell – Elemente zur Spezifikation der Eingangsgrößen eines Simulationsteilnehmers 149

Abbildung 74: Metamodell – Behaviour 150

Abbildung 75: Metamodell – *SimulationBehaviour* 150

Abbildung 76: Metamodell – *ProxyBehaviour* 150

Abbildung 77: Metamodell – *Event* (vgl. *Ereignis* in Abbildung 47) 151

Abbildung 78: Metamodell – *Action*, *Goal* und *Preference* (vgl. *Aktion*, *Ziel* und *Wert* in Abbildung 47) 151

Abbildung 79: Metamodell – *Limitation* (vgl. *Einschränkung* in Abbildung 47) 152

Abbildung 80: Metamodell – *Scenario* als Zusammenfassung der Inhalte einer Szenarioinstanz 152

Abbildung 81: Abstrahierte Veranschaulichung des Unmarshallings einer Szenarioinstanz (1) unter Nutzung der referenzierten Modellbibliothek (2) inklusive der Überführung in HLA-konforme Federates (3). 153

Abbildung 82: Integration eines Adapters als Vermittlerschicht zur Kapselung und Automatisierung der Kommunikation der *Federates* mit der zentralen *Runtime Infrastructure (RTI)* 156

Abbildung 83: Detailentwurf des RTI-Adapter zur Kapselung und Automatisierung der Kommunikation 157

Abbildung 84: Zentrale Zustandshaltung einer *SimulationFederate*-Instanz 157

Abbildung 85: Ghosting von Objekten anderer Simulationsteilnehmer 159

Abbildung 86: Ghosting von Objekten anderer Simulationsteilnehmer innerhalb einer kooperativen Simulation 159

Abbildung 87: Komponentenhierarchie eines Akteurs (links) und der entsprechende Verhaltensgraph (rechts) 160

Abbildung 88: Struktur und Beziehungen des *BehaviourGraph* eines *Actors* 160

Abbildung 89: UML-Aktivitätsdiagramm der Kommunikation und Übersetzung durch ein Proxy-Verhalten..... 162

Abbildung 90: UML-Aktivitätsdiagramm des Ablaufs der Konvertierung von Werten, bevor diese ein externes Textobjekt kommuniziert werden, entsprechend der als Teil des Szenarios spezifizierten *OutputTranslation*s. 163

Abbildung 91: Ablauf der Container-basierten Orchestrierung der verteilten Simulation 165

Abbildung 92: Repository-Struktur der öffentlich zugänglichen Ergebnisse..... 166

Abbildung 93: Dezentrale (links) und zentrale (rechts) Implementierung einer RTI..... 168

Abbildung 94: Jeweils eine eigenständige Instanz der lokalen RTI-Implementierung Porticos (LRTI) und der entwickelten *Model-Aware-RTI-Exchange* (MA-RTI-X) Adapterschicht ist Teil jedes Federates. 169

Abbildung 95: Lebenszyklus eines *SimulationFederate* als Container eines *Reactors*..... 170

Abbildung 96: Paketstruktur der Implementierung des Metamodells..... 171

Abbildung 97: Realisierung des hierarchischen Modells durch Abhängigkeiten zwischen Maven-Modulen 172

Abbildung 98: Kontrollfluss und Beziehungen der an der Interpretation einer Szenarioinstanz beteiligten Objekte ... 174

Abbildung 99: Portico WAN-Router als zentraler Kommunikationsknoten bei verteilter Ausführung..... 175

Abbildung 100: Elemente und Aktivitäten der mehrstufigen Evaluation (nach [Oppermann et al., 2019])..... 176

Abbildung 101: Screenshots von der Benutzeroberfläche der umgesetzten einfachen Webanwendung *WebView* zur Darstellung von durch den im Szenario konfigurierten *GlobalObserver* per *Websocket* gesendeten Daten während eines Simulationslaufs. Links: Zustand des Simulationslaufs auf Basis der Szenarioinstanz aus *Modellzweck 1* zum Zeitpunkt *t = 1*. Rechts: Zustand desselben Simulationslaufs zum Zeitpunkt *t = 141*..... 182

Abbildung 102: Screenshots der *WebView* während der für *Modellzweck 2* durchgeführten Simulationsläufe. Links: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne Hindernisse und andere Verkehrsteilnehmer zum Simulationszeitpunkt *t = 172*. Rechts: Zustand des Simulationslaufs auf Basis der um Hindernisse und weitere (passive) Verkehrsteilnehmer erweiterten Szenarioinstanz zum Simulationszeitpunkt *t = 184*. Das blaue Rechteck repräsentiert das Element *Waterbody*. Die roten und grünen Kegel repräsentieren *LateralMark*-Elemente des Typs *PORT_HAND* und *STARBOARD_HAND*. Die pinkfarbenen Kreise stellen die zurückgelegte Strecke des aktiven Schiffes dar. 183

Abbildung 103: Komponenten der umgesetzten Wirkstruktur. Auflistend dargestellt sind pro Komponente die individuelle *BehaviourTask*-Konkretisierungen, die dem Verhalten dieser Komponente zugeordnet sind..... 184

Abbildung 104: Screenshots der *WebView* während der für *Modellzweck 3* durchgeführten Simulationsläufe. Links: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne Hindernisse und andere Verkehrsteilnehmer zum Simulationszeitpunkt *t = 72*. Rechts: Zustand des Simulationslaufs auf Basis der um (passive) Verkehrsteilnehmer erweiterten Szenarioinstanz zum Simulationszeitpunkt *t = 72*. 185

Abbildung 105: Screenshots der *WebView* während zwei zur Erprobung der Interoperabilität durchgeführten Simulationsläufe. Oben: Zustand des Simulationslaufs auf Basis einer Szenarioinstanz ohne integrierte externe Testobjekte zum Simulationszeitpunkt *t=600*. Unten: Zustand eines Simulationslaufs auf Basis der um die Integration eines externen Testobjekts, welches den Wert der Geschwindigkeit linear erhöht, erweiterten Szenarioinstanz zum Simulationszeitpunkt *t=450*. 188

Abbildung 106: Zum Zweck der Untersuchung der Interoperabilität erprobte Verteilungsvarianten 189

Abbildung 107: Fallstudie – Erprobung einer Implementierung der MTCAS-Verhaltenskomponente..... 189

Abbildung 108: Fallstudie – Anbindung der MTCAS-Implementierung der *eMIR*-Plattform als externes Testobjekt. 190

Abbildung 109: Ausschnitt des für Modells die Durchführung der Fallstudie umgesetzten Modells..... 190

Abbildung 110: Detaillierter Versuchsaufbau der Fallstudie 191

Abbildung 111: Screenshots der *WebView* während eines Simulationslaufs der beschriebenen Fallstudie. Das von links kommende Schiff repräsentiert die von der MTCAS-Implementierung gelieferten Daten. Das von rechts kommende Schiff wird durch DisCo-BaTS simuliert und seine Positionsdaten werden an die externe MTCAS-Implementierung kommuniziert. Oben: Zustand des Simulationslaufs zum Simulationszeitpunkt *t = 150*; Mitte: Zustand des Simulationslaufs zum Simulationszeitpunkt *t = 475*; Unten: Zustand des Simulationslaufs zum Simulationszeitpunkt *t = 1100*..... 192

Abbildung 112: Screenshots der *WebView* während eines Simulationslaufs der beschriebenen Fallstudie zum Simulationszeitpunkt *t = 1025*. Das von links kommende Schiff repräsentiert die von der MTCAS-Implementierung gelieferten Daten. Das zugrundeliegende Szenario umfasst keine eigenständige Instanz eines Schiff-Elements. Da somit kein Bewegungsdaten erzeugt und an die externe MTCAS-Implementierung geliefert werden, leitet diese folgerichtig kein Ausweichmanöver ein. 192

Abbildung 113: Integration des erarbeiteten Lösungsansatzes in den *Distributed Simulation Engineering and Execution Process* (DSEEP) zur Umsetzung und Nutzung einer *High Level Architecture* (HLA) Föderation. 193

Abbildung 114: Erprobung eines Testobjekts bei gleichzeitiger Einspielung externer Verkehrsdaten 203

Abbildung 115: Mögliche Einspielung externer Infrastrukturdaten 203

TABELLEN

Tabelle 1: Automatisierungsgrade von Straßenfahrzeugen nach [SAE, J3016 APR2021] 29
Tabelle 2: Automatisierungsgrade von Luftfahrzeugen nach [EASA, 2023] 30
Tabelle 3: Autonomiegrade von Schiffen nach [IMO, 2021b]. 30
Tabelle 4: Skala des Grades der Automatisierung von Wasserfahrzeugen nach [One Sea, 2022]. 33
Tabelle 5: Betrachtungsebene des Simulationsmodells (Zeilen), die Art der möglichen Integration des jeweiligen Testobjekts in einen Simulationslauf (Spalten) und die Art von geeigneten Testobjekten (Zellen) 80
Tabelle 6: Eigenschaften und Beispiele funktionaler, logischer und konkreter Szenariospezifikationen 83
Tabelle 7: Merkmale der durchgeführten Studie nach [Cooper, 1988]. 94
Tabelle 8: Für die Literatursuche verwendete Schlüsselwörter und die thematischen Merkmale aus denen diese abgeleitet wurden. Da die genutzten Datenbanken die Meta-Daten zu den indexierten Veröffentlichungen in englischer Sprache pflegen, wurden auch die Schlüsselwörter in englischer Sprache verfasst und genutzt 95
Tabelle 9: Anzahl der gefundenen Datensätze pro durchsuchter Datenbank 97
Tabelle 10: Statistik des mehrstufigen Selektionsprozesses 97
Tabelle 11: Konzept-Matrix nach [Webster und Watson, 2002]. 98
Tabelle 12: Erfüllung der in Abschnitt 9 formulierten Anforderungen durch die identifizierten, zu Testzwecken eingesetzten maritimen Simulationsanwendungen. 99
Tabelle 13: Abbildung der Elemente der Wirkbeziehung realer Fahrzeuge durch die Elemente des Metamodells 128
Tabelle 14: Metamodell – kombinierte Ausprägungen der physischen und verhaltensbezogenen Charakteristiken 146
Tabelle 15: Merkmale verschiedener RTI-Implementierungen 169

LISTINGS

Listing 1: OMT-konforme XML-Repräsentation der Modellelemente (gekürzt) 155
Listing 2: OMT-konforme XML-Repräsentation einer *Property* durch jeweils drei Attribute (gekürzt) 155
Listing 3: Ausschnitt eines während des Packaging der Bibliothek erzeugten Klassen-Index 172
Listing 4: Ausschnitt eines während des Packaging der Bibliothek erzeugten Klassen-Index 173
Listing 5: CMD-Skript zum einfachen Start der Simulationsanwendung 173
Listing 6: Konfiguration der Verteilung in der XML-Spezifikation einer Szenarioinstanz (gekürzt) 174
Listing 7: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs 184
Listing 8: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs 186
Listing 9: Spezifikation der für *Modellzweck 3* beschriebene Komponentenstruktur des aktiven Akteurs 186
Listing 10: Ausschnitt der Konfiguration eines *ProxyBehaviours* in der XML-Repräsentation einer Szenarioinstanz .. 187

VERÖFFENTLICHUNGEN

Während der Arbeit an der vorliegenden Dissertation sind Beiträge in verschiedenen wissenschaftlichen Tagungsbänden und Fachzeitschriften veröffentlicht worden, die konzeptuelle Überschneidungen mit dieser Arbeit aufweisen und/oder Abbildungen ähnlich der hier gezeigten umfassen. Alle Einreichungen unterlagen dabei einem wissenschaftlichen Peer-Review-Verfahren. Die entsprechenden Veröffentlichungen sind nachfolgend, geordnet nach ihrem jeweiligen Veröffentlichungsdatum in aufsteigender Reihenfolge, aufgelistet.

Reiher, D. & Hahn, A. (2021): Towards a Model-Based Multi-Layered Approach to Describe Traffic Scenarios on a Technical Level. *Journal of Marine Science and Engineering (JMSE)*, 9 (6). DOI: 10.3390/jmse9060673

Reiher, D. & Hahn, A. (2021): Review on the Current State of Scenario- and Simulation-Based V&V in Application for Maritime Traffic Systems. In *OCEANS 2021: San Diego – Porto:* IEEE. DOI: 10.23919/OCEANS44145.2021.9705781

Reiher, D. & Hahn, A. (2023): Ad Hoc HLA Simulation Model Derived from a Model-Based Traffic Scenario. *SIMULATION.* Sage Publishing, 99 (8), S. 859-882. DOI: 10.1177/00375497231152651

Hake, G., Reiher, D., Mentjes, J. & Hahn, A. (2024): Integrating scenario- and contract-based verification for automated vessels. *Journal of Marine Science and Technology*. DOI: 10.1007/s00773-024-01008-0

ANHANG

A BEISPIELHAFTER RUMPF EINER SZENARIOINSTANZ

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <configuration>
    <runs>1</runs>
    <duration>0</duration>
    <library>
      <artifact>  </artifact>
      <suffix>  </suffix>
      <version>  </version>
      <group>  </group>
    </library>
    <distribution>
      <targets>
        <ssh alias=" ">
          <address>  </address>
          <username>  </username>
          <password>  </password>
        </ssh>
      </targets>
      <assignments>
        <assignment>
          <id>  </id>
          <targetMachine>  </targetMachine>
        </assignment>
      </assignments>
    </distribution>
    <output>
      <observer timeStepSize="1">
        <observedTypes>
          <observedType>
            <name>  </name>
            <properties...> <!-- list of <property>PROPERTY.NAME.PATH</property> nodes -->
          </observedType>
        </observedTypes>
        <outputConfigs...> <!-- list of <outputConfig xsi:type="CLASSNAME" /> nodes -->
      </observer>
    </output>
  </configuration>
  <scene> <!-- contents of the starting scene of this scenario instance -->
    <scenery> <!-- list of prop elements -->
      <prop xsi:type="lateralMark">
        <id>  </id>
        <name>  </name>
        <position>
          <value xsi:type="position">
            <x xsi:type="xs:double">  </x>
            <y xsi:type="xs:double">  </y>
            <z xsi:type="xs:double">  </z>
          </value>
        </position>
      </prop>
    </scenery>
    <actors> <!-- list of actor elements -->
      <actor xsi:type=" " timeStepSize="1">
        <id>  </id>
        <name>  </name>
        <position>
          <value xsi:type="position">
            <x xsi:type="xs:double">  </x>
            <y xsi:type="xs:double">  </y>
            <z xsi:type="xs:double">  </z>
          </value>
        </position>
        <behaviour xsi:type="  " />
      </actor>
    </actors>
    <limitations...> <!-- list of limitation elements -->
  </scene>
  <events...> <!-- list of event elements -->
  <actions...> <!-- list of action elements -->
  <goals...> <!-- list of goal elements -->
  <preferences...> <!-- list of preference elements -->
</scenario>
```

B SZENARIOINSTANZ: MODELLZWECK 1

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <configuration>
    <runs>1</runs>
    <duration>0</duration>
    <library>
      <artifact>example-eval1</artifact>
      <suffix>-jar-with-dependencies</suffix>
      <version>0.1-SNAPSHOT</version>
      <group>de.uol.discobats.model.o0-testcase.libraries</group>
    </library>
    <output ... > <!-- output observer configuration isn't relevant here -->
  </configuration>

  <scene>
    <actors>
      <actor xsi:type="unspecifiedVessel" timeStepSize="1">
        <id>own1</id>
        <name>Own Ship 1</name>
        <rotation xsi:type="double">0.0</rotation>
        <speed>
          <value xsi:type="xs:double">2.0</value>
          <unit>SPEED.KNOTS</unit>
          <prefix>NONE</prefix>
        </speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.5</latitude>
            <longitude xsi:type="double">4.0</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="simpleStraightlineDrivingBehaviour"/>
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <properties>
              <property>position</property>
            </properties>
          </observedType>
        </observedTypes>
      </actor>
      <actor xsi:type="unspecifiedVessel" timeStepSize="1">
        <id>ref1</id>
        <name>Reference Vessel 1</name>
        <rotation xsi:type="double">350</rotation>
        <speed xsi:type="double">0</speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.501111</latitude>
            <longitude xsi:type="double">4.000278</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="simpleStraightlineDrivingBehaviour"/>
      </actor>
      <actor xsi:type="unspecifiedVessel" timeStepSize="1">
        <id>ref2</id>
        <name>Reference Vessel 2</name>
        <rotation xsi:type="double">10</rotation>
        <speed xsi:type="double">0</speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.501111</latitude>
            <longitude xsi:type="double">4.000665</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="simpleStraightlineDrivingBehaviour"/>
      </actor>
    </actors>
  </scene>
</scenario>
```

C SZENARIOINSTANZ: MODELLZWECK 2

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <configuration>
    <runs>1</runs>
    <duration>0</duration>

    <library>
      <artifact>example-eval2</artifact>
      <suffix>-jar-with-dependencies</suffix>
      <version>0.1-SNAPSHOT</version>
      <group>de.uol.discobats.model.o0-testcase.libraries</group>
    </library>

    <output>
      <observer timeStepSize="4">
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <properties>
              <property>position</property>
              <property>name</property>
              <property>speed</property>
              <property>rotation</property>
            </properties>
          </observedType>
          <observedType>
            <name>lateralMark</name>
            <properties>
              <property>position</property>
              <property>name</property>
              <property>markType</property>
              <property>region</property>
            </properties>
          </observedType>
          <observedType>
            <name>waterbody</name>
            <properties>
              <property>width</property>
              <property>length</property>
              <property>position</property>
            </properties>
          </observedType>
        </observedTypes>
        <outputConfigs>
          <outputConfig xsi:type="webSocketOutputConfig">
            <address>localhost</address>
            <port>3001</port>
          </outputConfig>
        </outputConfigs>
      </observer>
    </output>
  </configuration>

  <scene>
    <actors>
      <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
        <id>own1</id>
        <name>Own Ship 1</name>
        <rotation xsi:type="double">0.0</rotation>
        <speed>
          <value xsi:type="xs:double">2.0</value>
          <unit>SPEED.KNOTS</unit>
          <prefix>NONE</prefix>
        </speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.4995</latitude>
            <longitude xsi:type="double">3.9997</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="simpleFollowRouteAvoidanceBehaviour"/>
```

```
      <observedTypes>
          <name>vessel</name>
          <properties>
            <property>position</property>
            <property>rotation</property>
          </properties>
        </observedType>
        <observedType>
          <name>lateralMark</name>
          <properties>
            <property>position</property>
            <property>markType</property>
            <propert>color</propert>
          </properties>
        </observedType>
      </observedTypes>
    </actor>

    <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
      <id>ref1</id>
      <name>Reference Vessel 1</name>
      <rotation xsi:type="double">350</rotation>
      <speed xsi:type="double">0</speed>
      <position>
        <value xsi:type="position">
          <latitude xsi:type="double">54.501011</latitude>
          <longitude xsi:type="double">4.000278</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </value>
      </position>
      <behaviour xsi:type="nullBehaviour"/>
    </actor>
    <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
      <id>ref2</id>
      <name>Reference Vessel 2</name>
      <rotation xsi:type="double">10</rotation>
      <speed xsi:type="double">0</speed>
      <position>
        <value xsi:type="position">
          <latitude xsi:type="double">54.5011</latitude>
          <longitude xsi:type="double">4.000665</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </value>
      </position>
      <behaviour xsi:type="nullBehaviour"/>
    </actor>
  </actors>

  <scenery>
    <prop xsi:type="waterbody">
      <id>area1</id>
      <name>Waterbody 1</name>
      <width xsi:type="double">125</width>
      <length xsi:type="double">325</length>
      <rotation>0</rotation>
      <position>
        <value xsi:type="position">
          <latitude xsi:type="double">54.501005</latitude>
          <longitude xsi:type="double">4.000278</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </value>
      </position>
      <usableBy>
        <domain xsi:type="trafficDomain">MARITIME</domain>
      </usableBy>
    </prop>
    <!-- PORT HAND (RED) -->
    <prop xsi:type="lateralMark">
      <id>mark1</id>
      <name>Lateral Port 1</name>
      <rotation>0</rotation>
      <position>
        <value xsi:type="position">
          <latitude xsi:type="double">54.50125</latitude>
          <longitude xsi:type="double">4.000278</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </value>
      </position>
      <markType>
```

```
          <value xsi:type="lateralMarkType">PORT_HAND</value>
        </markType>
        <region>
          <value xsi:type="region">REGION_A</value>
        </region>
        <number>lateralMark1</number>
      </prop>
      <prop xsi:type="lateralMark">
        <id>mark2</id>
        <name>Lateral Port 2</name>
        <rotation>0</rotation>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.5015</latitude>
            <longitude xsi:type="double">4.000278</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <markType>
          <value xsi:type="lateralMarkType">PORT_HAND</value>
        </markType>
        <region>
          <value xsi:type="region">REGION_A</value>
        </region>
        <number>lateralMark2</number>
      </prop>
      <prop xsi:type="lateralMark">
        <id>mark3</id>
        <name>Lateral Port 3</name>
        <rotation>0</rotation>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.50175</latitude>
            <longitude xsi:type="double">4.000278</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <markType>
          <value xsi:type="lateralMarkType">PORT_HAND</value>
        </markType>
        <region>
          <value xsi:type="region">REGION_A</value>
        </region>
        <number>lateralMark3</number>
      </prop>
      <!-- STARBOARD HAND (GREEN) -->
      <prop xsi:type="lateralMark">
        <id>mark4</id>
        <name>Lateral Mark Star 1</name>
        <rotation>0</rotation>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.5003</latitude>
            <longitude xsi:type="double">4.00005</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <markType>
          <value xsi:type="lateralMarkType">STARBOARD_HAND</value>
        </markType>
        <region>
          <value xsi:type="region">REGION_A</value>
        </region>
        <number>lateralMark3</number>
      </prop>
    </scenery>
  </scene>

  <goals>
    <goal xsi:type="routeGoal">
      <targetActorId>own1</targetActorId>
      <route>
        <position>
          <latitude xsi:type="double">54.50235</latitude>
          <longitude xsi:type="double">4.0006</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </position>
      </route>
    </goal>
  </goals>
</scenario>
```

D SZENARIOINSTANZ: MODELLZWECK 3

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <configuration>
    <runs>1</runs>
    <duration>0</duration>

    <library>
      <artifact>example-eval3</artifact>
      <suffix>-jar-with-dependencies</suffix>
      <version>0.1-SNAPSHOT</version>
      <group>de.uol.discobats.model.o0-testcase.libraries</group>
    </library>

    <output>
      <observer timeStepSize="4">
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <properties>
              <property>position</property>
              <property>name</property>
              <property>rotation</property>
              <property>speed</property>
            </properties>
          </observedType>
        </observedTypes>
        <outputConfigs>
          <outputConfig xsi:type="webSocketOutputConfig">
            <address>localhost</address>
            <port>3001</port>
          </outputConfig>
        </outputConfigs>
      </observer>
    </output>
  </configuration>

  <scene>
    <actors>
      <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
        <id>own1</id>
        <name>Own Ship 1</name>
        <rotation xsi:type="double">0.0</rotation>
        <mmsi xsi:type="string">211123456</mmsi>
        <speed>
          <value xsi:type="xs:double">2.0</value>
          <unit>SPEED.KNOTS</unit>
          <prefix>NONE</prefix>
        </speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.500552</latitude>
            <longitude xsi:type="double">4.000263</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="nullBehaviour"/>
        <components>
          <component xsi:type="captain" timeStepSize="1">
            <executionGroup>2</executionGroup>
          </component>
          <component xsi:type="sensorDataProcessor" timeStepSize="1">
            <executionGroup>1</executionGroup>
            <components>
              <component xsi:type="sensorAIS" timeStepSize="1">
                <executionGroup>1</executionGroup>
              </component>
              <component xsi:type="sensorGPS" timeStepSize="1">
                <executionGroup>1</executionGroup>
              </component>
            </components>
          </component>
        </components>
```

```
        <observedTypes>
          <observedType>
            <name>seagoingVessel</name>
            <properties>
              <property>name</property>
              <property>position</property>
              <property>rotation</property>
              <property>mmsi</property>
            </properties>
          </observedType>
        </observedTypes>
      </actor>

      <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
        <id>ref1</id>
        <name>Reference Vessel 1</name>
        <rotation xsi:type="double">350</rotation>
        <speed xsi:type="double">0</speed>
        <mmsi xsi:type="string">211123457</mmsi>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.501011</latitude>
            <longitude xsi:type="double">4.000278</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="nullBehaviour"/>
      </actor>

      <actor xsi:type="unspecifiedSeagoingVessel" timeStepSize="1">
        <id>ref2</id>
        <name>Reference Vessel 2</name>
        <mmsi xsi:type="string">211123458</mmsi>
        <rotation xsi:type="double">10</rotation>
        <speed xsi:type="double">0</speed>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="double">54.5011</latitude>
            <longitude xsi:type="double">4.000665</longitude>
            <altitude xsi:type="double">0.0</altitude>
          </value>
        </position>
        <behaviour xsi:type="nullBehaviour"/>
      </actor>

    </actors>

    <scenery>
      <!-- empty scenery -->
    </scenery>
  </scene>

  <goals>
    <goal xsi:type="routeGoal">
      <targetActorId>own1</targetActorId>
      <route>
        <position>
          <latitude xsi:type="double">54.501745</latitude>
          <longitude xsi:type="double">4.000204</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </position>
      </route>
    </goal>
  </goals>

</scenario>
```

E SZENARIOINSTANZ: ERPROBUNG DER INTEROBERABILITÄT

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <configuration>
    <runs>1</runs>
    <duration>0</duration>

    <library>
      <artifact>example</artifact>
      <suffix>-jar-with-dependencies</suffix>
      <version>0.1-SNAPSHOT</version>
      <group>de.uol.discobats.model.o0-testcase.libraries</group>
    </library>

    <distribution>
      <targets>
        <target xsi:type="sshTarget">
          <id>rpi</id>
          <address>IP-ADDRESS/FQL-HOSTNAME</address>
          <username>USERNAME</username>
          <password>PASSWORD</password>
        </target>
      </targets>
      <assignments>
        <assignment>
          <target>rpi</target>
          <actor>external</actor>
        </assignment>
      </assignments>
    </distribution>

    <output>
      <observer timeStepSize="25">
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <properties>
              <property>position</property>
              <property>name</property>
              <property>speed</property>
              <property>rotation</property>
            </properties>
          </observedType>
        </observedTypes>
        <outputConfigs>
          <outputConfig xsi:type="webSocketOutputConfig">
            <address>localhost</address>
            <port>3001</port>
          </outputConfig>
        </outputConfigs>
      </observer>
    </output>
  </configuration>

  <scene>
    <actors>
      <!-- PROXY ACTOR -->
      <actor xsi:type="exampleContainerShip" timeStepSize="1">
        <id>external</id>
        <name>External 1</name>
        <imo xsi:type="string">9461052</imo>
        <mmsi xsi:type="string">218774001</mmsi>
        <callsign>DFKM3</callsign>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="xs:double">53.912832</latitude>
            <longitude xsi:type="xs:double">7.861748</longitude>
            <altitude xsi:type="xs:double">0.0</altitude>
          </value>
        </position>
        <rotation xsi:type="xs:double">135.0</rotation>
        <formString xsi:type="xs:double">POLYGON ((0 0, 48 0, 48 366, 0 366, 0 0))</formString>
        <acceleration xsi:type="xs:double">0</acceleration>
        <behaviour xsi:type="observingSimpleFollowRouteBehaviour"/>
```

```
<course xsi:type="xs:double">135.0</course>
        <weight xsi:type="xs:double">66850</weight>
        <components>
          <component xsi:type="examplePlainComponent">
            <behaviour xsi:type="proxyBehaviour">
              <communicationAdapter xsi:type="exampleUDPCommunicationAdapter">
                <address>IP-ADDRESS</address>
                <remoteInPort>1337</remoteInPort>
                <localOutPort>1338</localOutPort>
                <localInPort>1339</localInPort>
              </communicationAdapter>
              <formatAdapter xsi:type="exampleXMLFormatAdapter"/>
              <dictionary>
                <externalValues>
                  <value>
                    <name>in_own_speed</name>
                    <type>double</type>
                    <path>ownship.status.speed</path>
                  </value>
                  <value>
                    <name>out_own_speed</name>
                    <path>ownship.speed</path>
                    <type>double</type>
                    <unit>SPEED.SPEED_OF_LIGHT</unit>
                    <prefix>NONE</prefix>
                  </value>
                  <value>
                    <name>out_own_name</name>
                    <path>ownship.name</path>
                  </value>
                  <value>
                    <name>out_own_weight</name>
                    <path>ownship.weight</path>
                    <type>double</type>
                  </value>
                  <value>
                    <name>out_own_latitude</name>
                    <path>ownship.position.latitude</path>
                  </value>
                  <value>
                    <name>out_own_longitude</name>
                    <path>ownship.position.longitude</path>
                  </value>
                  <value>
                    <name>out_others_weight</name>
                    <type>double</type>
                    <path>others[].weight</path>
                    <unit>MASS.GRAM</unit>
                    <prefix>KILO</prefix>
                  </value>
                </externalValues>
                <input>
                  <translation>
                    <internalPath>self.main.speed</internalPath>
                    <externalValue>in_own_speed</externalValue>
                  </translation>
                </input>
                <output>
                  <translation>
                    <internalPath>self.main.speed</internalPath>
                    <externalValue>out_own_speed</externalValue>
                  </translation>
                  <translation>
                    <internalPath>self.main.name</internalPath>
                    <externalValue>out_own_name</externalValue>
                  </translation>
                  <translation>
                    <internalPath>self.main.weight</internalPath>
                    <externalValue>out_own_weight</externalValue>
                  </translation>
                  <translation>
                    <internalPath>self.main.position.longitude</internalPath>
                    <externalValue>out_own_longitude</externalValue>
                  </translation>
                  <translation>
                    <internalPath>self.main.position.latitude</internalPath>
                    <externalValue>out_own_latitude</externalValue>
                  </translation>
                  <translation>
```

```
                    <internalPath>env.vessel[].weight</internalPath>
                    <externalValue>out_others_weight</externalValue>
                  </translation>
                </output>
              </dictionary>
            </behaviour>
          </component>
        </components>
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <attributes>
              <attribute>name</attribute>
              <attribute>speed</attribute>
              <attribute>position</attribute>
              <attribute>weight</attribute>
            </attributes>
          </observedType>
        </observedTypes>
      </actor>

      <!-- FULLY SIMULATED ACTOR -->
      <actor xsi:type="exampleContainerShip" timeStepSize="1">
        <id>internal</id>
        <name>Internal 1</name>
        <imo xsi:type="string">9461051</imo>
        <mmsi xsi:type="string">218774000</mmsi>
        <callsign>DFKM2</callsign>
        <position>
          <value xsi:type="position">
            <latitude xsi:type="xs:double">53.85151</latitude>
            <longitude xsi:type="xs:double">8.02517</longitude>
            <altitude xsi:type="xs:double">0.0</altitude>
          </value>
        </position>
        <rotation xsi:type="xs:double">135.0</rotation>
        <formString xsi:type="xs:double">POLYGON ((0 0, 48 0, 48 366, 0 366, 0 0))</formString>
        <acceleration xsi:type="xs:double">0</acceleration>
        <behaviour xsi:type="simpleFollowRouteBehaviour"/>
        <course xsi:type="xs:double">135.0</course>
        <weight xsi:type="xs:double">66850</weight>
      </actor>
    </actors>
    <scenery>
      <!-- empty scenery -->
    </scenery>
  </scene>

  <goals>
    <goal xsi:type="routeGoal">
      <targetActorId>external</targetActorId>
      <route>
        <position>
          <latitude xsi:type="double">53.903529</latitude>
          <longitude xsi:type="double">7.566147</longitude>
          <altitude xsi:type="double">0.0</altitude>
        </position>
      </route>
    </goal>
    <goal xsi:type="routeGoal">
      <targetActorId>internal</targetActorId>
      <route>
        <position>
          <latitude xsi:type="xs:double">54.00200</latitude>
          <longitude xsi:type="xs:double">7.65012</longitude>
          <altitude xsi:type="xs:double">0.0</altitude>
        </position>
        <position>
          <latitude xsi:type="xs:double">53.92355</latitude>
          <longitude xsi:type="xs:double">7.73266</longitude>
          <altitude xsi:type="xs:double">0.0</altitude>
        </position>
        <position>
          <latitude xsi:type="xs:double">53.98416</latitude>
          <longitude xsi:type="xs:double">8.23940</longitude>
          <altitude xsi:type="xs:double">0.0</altitude>
        </position>
      </route>
    </goal>
  </goals>
</scenario>
```

F SZENARIOINSTANZ: FALLSTUDIE

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<scenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <configuration>
    <runs>1</runs>
    <duration>0</duration>

    <library>
      <artifact>example</artifact>
      <suffix>-jar-with-dependencies</suffix>
      <version>0.1-SNAPSHOT</version>
      <group>de.uol.discobats.model.o0-testcase.libraries</group>
    </library>

    <distribution>
      <targets>
        <target xsi:type="sshTarget">
          <id>rpi</id>
          <address>IP-ADDRESS/FQL-HOSTNAME</address>
          <username>USERNAME</username>
          <password>PASSWORD</password>
        </target>
      </targets>
      <assignments>
        <assignment>
          <target>rpi</target>
          <actor>ship_ext_mtcas</actor>
        </assignment>
        <assignment>
          <target>rpi</target>
          <actor>ship_ext_target</actor>
        </assignment>
      </assignments>
    </distribution>

    <output>
      <observer timeStepSize="25">
        <observedTypes>
          <observedType>
            <name>vessel</name>
            <properties>
              <property>position</property>
              <property>name</property>
              <property>speed</property>
              <property>rotation</property>
            </properties>
          </observedType>
        </observedTypes>
        <outputConfigs>
          <outputConfig xsi:type="webSocketOutputConfig">
            <address>localhost</address>
            <port>3001</port>
          </outputConfig>
        </outputConfigs>
      </observer>
    </output>

  </configuration>

  <scene>
    <scenery>
      <!-- empty scenery -->
    </scenery>
    <actors>
      <!-- PROXIED MAIN BEHAVIOUR (GET POSITION OF MTCAS-OWNSHIP) -->
      <actor xsi:type="exampleContainerShip" timeStepSize="1">
        <id>ship_ext_mtcas</id>
        <name>External MTCAS</name>
        <behaviour xsi:type="proxyBehaviour">
          <mode>SYNC</mode>
          <inputCommunicationAdapter xsi:type="exampleUDPCommunicationAdapter">
            <localInPort>1339</localInPort>
          </inputCommunicationAdapter>
          <inputFormatAdapter xsi:type="exampleEMirXMLFormatAdapter"/>
```

```
        <dictionary>
          <externalValues>
            <value>
              <name>name</name>
              <type>string</type>
              <path>ComboundTaskProvider.provider[name=Ownship].name</path>
            </value>
            <value>
              <name>mmsi</name>
              <type>string</type>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.mmsi</path>
            </value>
            <value>
              <name>imo</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.imo</path>
              <type>string</type>
            </value>
            <value>
              <name>callsign</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.callSign</path>
              <type>string</type>
            </value>
            <value>
              <name>orientation</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.pose.orientation.z.value</path>
              <type>double</type>
              <unit>ANGLE.DEGREE</unit>
            </value>
            <value>
              <name>longitude</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.pose.coordinate.y</path>
              <type>double</type>
              <unit>ANGLE.DEGREE</unit>
            </value>
            <value>
              <name>latitude</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.pose.coordinate.x</path>
              <type>double</type>
              <unit>ANGLE.DEGREE</unit>
            </value>
            <value>
              <name>speed</name>
              <path>ComboundTaskProvider.provider[name=Ownship].vessel.characteristics[xsi:type=Dynamic
                    ObjectCharacteristic].linearVelocity.value.y</path>
              <type>double</type>
              <unit>SPEED.METERS_PER_SECOND</unit>
              <prefix>NONE</prefix>
            </value>
          </externalValues>
          <input>
            <translation>
              <internalPath>self.main.speed</internalPath>
              <externalValue>speed</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.position.latitude</internalPath>
              <externalValue>latitude</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.position.longitude</internalPath>
              <externalValue>longitude</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.rotation</internalPath>
              <externalValue>orientation</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.course</internalPath>
              <externalValue>orientation</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.mmsi</internalPath>
              <externalValue>mmsi</externalValue>
            </translation>
            <translation>
              <internalPath>self.main.callsign</internalPath>
              <externalValue>callsign</externalValue>
            </translation>
```

```
            <translation>
              <internalPath>self.main.imo</internalPath>
              <externalValue>imo</externalValue>
            </translation>
          </input>
        </dictionary>
      </behaviour>
    </actor>

    <!-- PROXIED COMPONENT BEHAVIOUR (SET POSITION OF MTCAS-TARGETSHIP) -->
    <actor xsi:type="exampleContainerShip" timeStepSize="2">
      <id>ship_ext_target</id>
      <name>External Target</name>
      <behaviour xsi:type="simpleFollowRouteBehaviour"/>
      <components>
        <component xsi:type="examplePlainComponent">
          <behaviour xsi:type="proxyBehaviour">
            <mode>FREE</mode>
            <outputCommunicationAdapter xsi:type="exampleUDPCommunicationAdapter">
              <address>192.168.1.10</address>
              <remoteInPort>1337</remoteInPort>
              <localOutPort>1338</localOutPort>
            </outputCommunicationAdapter>
            <outputFormatAdapter xsi:type="exampleNMEA0183FormatAdapter"/>
            <dictionary>
              <externalValues>
                <value>
                  <name>latitude</name>
                  <path>position.latitude</path>
                </value>
                <value>
                  <name>longitude</name>
                  <path>position.longitude</path>
                </value>
              </externalValues>
              <output>
                <translation>
                  <internalPath>self.main.position.latitude</internalPath>
                  <externalValue>latitude</externalValue>
                </translation>
                <translation>
                  <internalPath>self.main.position.longitude</internalPath>
                  <externalValue>longitude</externalValue>
                </translation>
              </output>
            </dictionary>
          </behaviour>
        </component>
      </components>
      <position>
        <value xsi:type="position">
          <latitude xsi:type="xs:double">54</latitude>
          <longitude xsi:type="xs:double">8.0849075</longitude>
          <altitude xsi:type="xs:double">0.0</altitude>
        </value>
      </position>
      <speed xsi:type="xs:double">6</speed>
      <rotation xsi:type="xs:double">270</rotation>
      <course xsi:type="xs:double">270</course>
    </actor>

  </actors>
</scene>

<goals>
  <goal xsi:type="routeGoal">
    <targetActorId>ship_ext_target</targetActorId>
    <route>
      <position>
        <latitude xsi:type="xs:double">54.00414</latitude>
        <longitude xsi:type="xs:double">7.91325</longitude>
        <altitude xsi:type="xs:double">0.0</altitude>
      </position>
    </route>
  </goal>
</goals>

</scenario>
```

G DOCKERFILE FÜR REMOTE-FEDERATE IMAGES

```
# PROJECT: Distributed Component-Based Traffic Simulation (DisCo-BaTS)
# NAME:    REMOTE-FEDERATE CONTAINER-IMAGE
# AUTHOR:  David Reiher
#-------------------------------------

# use alpine specific build of eclipse temurin, as alpine ueses a different
# c compiler library and thus requires a changed jdk/jre runtime
#
# this may cause problems if specific c funtions are required and thus a
# fallback to debian using the original jdk may be required

# min version is 21 to have linux/arm64/v8 versions availablefor multiplatform builds (rpi support)
FROM eclipse-temurin:21-alpine AS jre-build

# create a custom java runtime environment (jre) by utilizing jlink
RUN "$JAVA_HOME"/bin/jlink \
      --add-modules java.base,java.desktop,java.instrument,java.management,java.naming,java.prefs, \
                    java.rmi,java.scripting,java.security.jgss,java.sql,jdk.compiler,jdk.unsupported \
      --strip-debug \
      --no-man-pages \
      --no-header-files \
      --compress=2 \
      --output /javaruntime

#-------------------------------------

# base image
FROM alpine:3.20 AS base-image

# set java env variables
ENV JAVA_HOME=/opt/java/openjdk
ENV PATH="${JAVA_HOME}/bin:${PATH}"

# use the previously created custom java runtime
COPY --from=jre-build /javaruntime $JAVA_HOME

# set up application and needed resources
ENV IN_DOCKER=true

RUN mkdir /opt/app
RUN mkdir /FOMS

#-------------------------------------

FROM base-image AS copy-build-resources

# use the paths given by command line command
ARG APP_CLIENT_PATH
ARG PROPERTIES_FOLDER_PATH
ARG MODEL_LIBRARY_PATH

# copy discobats specific files into the container
COPY ${APP_CLIENT_PATH} /opt/app/discobats-client.jar
COPY ${PROPERTIES_FOLDER_PATH} simulation/properties
COPY ${MODEL_LIBRARY_PATH} /opt/app/model_library.jar

# start the client simulation only allowing the IPv4 network stack
ENTRYPOINT ["sh", "-c", "java -Djava.net.preferIPv4Stack=true -javaagent:/opt/app/discobats-client.jar
                            -jar /opt/app/discobats-client.jar -lib /opt/app/model_library.jar
                            -s /config/scenario.xml -a"]

#-------------------------------------
```