Co-Simulation von zur Laufzeit erweiterbaren Automatisierungssystemen im Internet der Dinge

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.) genehmigte Abhandlung

Vorgelegt von
Tobias Jung
aus Reichenbach (Berglen)

Hauptberichter:	Prof. Dr.-Ing. Dr. h. c. Michael Weyrich
Mitberichter:	Prof. Dr.-Ing. habil. Leon Urbas
Tag der Einreichung:	22.09.2022
Tag der mündlichen Prüfung:	21.11.2023

Institut für Automatisierungstechnik und Softwaresysteme
der Universität Stuttgart

2022

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme (IAS) der Universität Stuttgart.

Mein besonderer Dank gilt meinem Doktorvater und Leiter des Instituts, Herrn Prof. Dr.-Ing. Dr. h. c. Michael Weyrich, für die Übernahme des Hauptberichts, die vielen konstruktiven Diskussionen und wertvollen Hinweise und Anregungen bei der Erstellung dieser Arbeit.

Herr Prof. Dr.-Ing. habil. Leon Urbas danke ich für das Interesse an meiner Arbeit und die Übernahme des Mitberichts.

Ein besonderer Dank gilt Herrn Dr.-Ing. Nasser Jazdi für seine fortwährende Unterstützung während der Arbeit und die vielen wertvollen Gespräche.

Allen Kolleginnen und Kollegen am IAS gilt mein herzlicher Dank für die gute Zusammenarbeit und die vielen guten und konstruktiven Diskussionen.

Ebenso gilt mein Dank den Studierenden, die im Rahmen ihrer Master-, Studien-, Forschungs- und Bachelorarbeiten einen Beitrag zum Gelingen dieser Arbeit geleistet haben.

Stuttgart, im September 2022

Tobias Jung

Inhaltsverzeichnis

Abbildungsverzeichnis ..... **iv**

Tabellenverzeichnis ..... **v**

Abkürzungsverzeichnis ..... **vi**

Zusammenfassung ..... **vii**

Abstract ..... **viii**

1. Einleitung **1**
1.1 Bedeutung von Simulation 1
1.2 Herausforderungen der Simulation von IoT-Systemen 4
1.3 Zielsetzung der Arbeit 6
1.4 Abgrenzung der Arbeit 8
1.5 Aufbau der Arbeit 9

2 Stand der Forschung und Technik **11**
2.1 Simulation 11
2.1.1 Bedeutung von Simulation 12
2.1.2 Klassifikation von Simulationsarten 13
2.1.3 Simulationsphasen 15
2.2 Internet der Dinge 16
2.2.1 Schlüsseltechnologien im IoT 17
2.2.2 Elementare Anwendungsfälle im IoT 18
2.2.3 Anwendungsbeispiel des IoT 19
2.3 Bestehende Ansätze zur Simulation von IoT-Systemen 20
2.3.1 Diskrete-Event-Simulatoren 21
2.3.2 Agentenbasierte Simulatoren 25
2.3.3 Zusammenfassung 30
2.4 Bestehende Ansätze zur dynamischen Co-Simulation 31
2.4.1 Co-Simulation 32
2.4.2 Co-Simulation mit domänenspezifischen Standards 33
2.4.3 Dynamische Co-Simulation mit Functional Mock-up Interface 33
2.4.4 Dynamische Co-Simulation mit High Level Architecture 34
2.4.5 Allgemeine Co-Simulationsansätze aus der Literatur 34
2.4.6 Dynamische Co-Simulation mit SOA 35
2.4.7 Dynamische Co-Simulation mit OSGi 35
2.4.8 Dynamische Co-Simulation mit Agenten 36
2.4.9 Bewertung der Co-Simulationsansätze 38
2.5 Zusammenfassende Bewertung der Ansätze und Folgerung 40

3 Konzept zur dynamischen Co-Simulation **42**
3.1 Grundlegende Entscheidungen und Konzeptübersicht 44

3.1.1 Entscheidung zur Einbindung von Simulationstools ... 44
3.1.2 Überblick über die Co-Simulation ... 46
3.1.3 Co-Simulationsumgebung und Simulationsvertreter ... 49
3.1.4 Anforderungen an die Teilsimulationen und ihre Simulationstools ... 51
3.2 Datenaustausch in Co-Simulationen ... 52
3.2.1 Datenaustausch bei Co-Simulationsstandards ... 52
3.2.2 Datenaustausch in der Co-Simulationsumgebung ... 54
3.2.3 Datenaustauschschnittstellen ... 55
3.2.4 Anforderungen an die Teilsimulationen und ihre Simulationstools ... 57
3.3 Interaktionssimulationen ... 57
3.3.1 Interaktionsarten im IoT ... 58
3.3.2 Kommunikationsorientierte Interaktionssimulation ... 61
3.3.3 Prozessorientierte Interaktionssimulation ... 69
3.3.4 Anforderungen an die Teilsimulationen und ihre Simulationstools ... 76
3.4 Synchronisation der Co-Simulation ... 78
3.4.1 Synchronisationsansätze ... 79
3.4.2 Synchronisation bei Co-Simulationsstandards ... 82
3.4.3 Bewertung der Synchronisationsansätze ... 83
3.4.4 Synchronisationskonzept ... 83
3.4.5 Taktgeber ... 83
3.4.6 Synchronisationsschnittstelle ... 84
3.4.7 Anforderungen an die Teilsimulationen und ihre Simulationstools ... 85
3.4.8 Erweiterung der Synchronisation ... 85

4 Realisierung der Co-Simulation als Framework ... 87
4.1 Konzept des Frameworks der Co-Simulationsumgebung und der Simulationsvertreter ... 88
4.1.1 Anforderungen an das Co-Simulationsumgebungsframework ... 88
4.1.2 Realisierungsmöglichkeiten ... 90
4.1.3 Diskussion der Realisierungsmöglichkeiten ... 92
4.2 Konzept zur Erstellung der Schnittstellen ... 93
4.2.1 Konzeption der Schnittstellen ... 93
4.2.2 Befähigung von zusätzlichen Simulationstools ... 97
4.2.3 Anbindung an FMI ... 98
4.3 Modellbibliothek zur Erstellung der Interaktionssimulationen ... 99
4.3.1 Kommunikationsorientierte Interaktionssimulation ... 100
4.3.2 Prozessorientierte Interaktionssimulation ... 100

5 Implementierung des Co-Simulationsframeworks ... 102
5.1 Implementierung des Co-Simulationsframeworks ... 102
5.1.1 Co-Simulationsumgebung ... 102
5.1.2 Simulationsvertreter ... 103
5.1.3 Benutzeroberfläche ... 104
5.2 Implementierung der Schnittstellen ... 104
5.2.1 Generische Schnittstellen ... 105
5.2.2 Schnittstellen zu Komponentensimulationen ... 105
5.2.3 Schnittstelle zur Kommunikationssimulation ... 107
5.2.4 Schnittstelle zur Prozesssimulation ... 107

5.3 Implementierung der Synchronisation ... 107
5.3.1 Implementierung des Taktgebers ... 107
5.3.2 Realisierung der Synchronisation in den einzelnen Simulationstools ... 108
5.4 Implementierung der Interaktionssimulationen ... 109
5.4.1 Kommunikationssimulation in OMNet++ ... 109
5.4.2 Prozessinteraktion in Unity 3D ... 110

6 Evaluierung der Co-Simulation ... 112
6.1 Beschreibung des Szenarios ... 112
6.1.1 Simuliertes IoT-Szenario ... 113
6.1.2 Modellbeschreibung ... 117
6.1.3 Ablauf des Evaluierungsszenarios und der Simulation ... 118
6.2 Erfüllung der Herausforderungen ... 119
6.3 Erfüllung der Zielsetzung ... 121

7 Schlussbetrachtungen ... 125
7.1 Zusammenfassung der Ergebnisse und Bewertung ... 125
7.2 Ausblick ... 127

Literaturverzeichnis ... 128

Anhang ... 139
Erweiterung durch Hardware-in-the-Loop ... 139
Beschreibung der Modelle ... 142

Begriffsverzeichnis ... 149

Abbildungsverzeichnis

Abbildung 1: Aufbau der Arbeit 10
Abbildung 2: Entwicklung der Simulation über die Zeit [41] 12
Abbildung 3: Co-Simulation mit Teilsimulationen 46
Abbildung 4: Co-Simulation mit Interaktionssimulation 48
Abbildung 5: Co-Simulation mit Taktgeber 49
Abbildung 6: Aufbau der Co-Simulationsumgebung und der Simulationsvertreter 50
Abbildung 7: Komponentensimulation als Co-Simulation 51
Abbildung 8: Datenaustausch in der Co-Simulationsumgebung 55
Abbildung 9: Konzept der Datenaustauschschnittstelle 56
Abbildung 10: Datenaustauschschnittstelle mit einer simulationstoolspezifischen Schnittstelle 57
Abbildung 11: Interaktionsarten im IoT 59
Abbildung 12: Co-Simulation mit Kommunikationssimulation und Prozesssimulation 59
Abbildung 13: Erweiterte Co-Simulationsumgebung 60
Abbildung 14: Erweiterte Datenaustauschschnittstelle 60
Abbildung 15: Weg einer Kommunikationsnachricht 62
Abbildung 16: Kommunikationsnachrichtenspezifikation 63
Abbildung 17: Kommunikationsorientierte Schnittstelle einer Kommunikationssimulation 65
Abbildung 18: Kommunikationsorientierte Schnittstelle einer Komponentensimulation 66
Abbildung 19: Co-Simulationsumgebung mit Kommunikationsvertreter 67
Abbildung 20: Prozessorientierter Interaktionsablauf 70
Abbildung 21: Co-Simulationsumgebung mit Prozessvertreter 75
Abbildung 22: Prozessorientierte Schnittstelle einer Prozesssimulation 75
Abbildung 23: Prozessorientierte Schnittstelle einer Komponentensimulation 76
Abbildung 24: Kausalitätsfehler 79
Abbildung 25: Konservative Synchronisation (Synchroner Betrieb) 80
Abbildung 26: Optimistische Synchronisation 81
Abbildung 27: Synchronisationsschnittstelle 84
Abbildung 28: Realisierung mit OSGi 90
Abbildung 29: Realisierung über eine serviceorientierte Architektur 91
Abbildung 30: Realisierung durch Agenten 92
Abbildung 31: Kommunikation mit einem Simulationstool über Sockets 96
Abbildung 32: Kommunikation mit einem Simulationstool über CORBA 97
Abbildung 33: Befähigung von zusätzlichen Simulationstools durch Nachrichten-Mapping 98
Abbildung 34: Anbindung von FMI über eine von FMI zur Verfügung gestellten API 99
Abbildung 35: Übersicht über die Implementierung 102
Abbildung 36: Benutzeroberfläche des Frameworks 104
Abbildung 37: Benutzeroberfläche zum Einbinden einer Teilsimulation 105
Abbildung 38: Kommunikationssimulation in OMNet++ 110
Abbildung 39: Prozesssimulation eines Warenlagers in Unity 3D 111
Abbildung 40: Überblick über das Evaluierungsszenario in der Prozesssimulation 114

Tabellenverzeichnis

Tabelle 1: Übersicht der Diskrete-Event-Simulationsansätze 24
Tabelle 2: Übersicht der agentenbasierten Simulationsansätze 29
Tabelle 3: Bewertung der Diskrete-Event- und agentenbasierten Simulatoren hinsichtlich der Anforderungen an eine Simulation eines IoT-Systems 31
Tabelle 4: Vergleich der Co-Simulationsansätze 40
Tabelle 5: Anpassungen zur kommunikationsorientierten Interaktionssimulation 61
Tabelle 6: Anpassungen zur Prozessorientierten Interaktionssimulation 69
Tabelle 7: Beispiele für prozessorientierte Dienste 71
Tabelle 8: Vergleich der Realisierungsmöglichkeiten 93
Tabelle 9: Simulationstools und die von ihnen verwendeten Schnittstellen 94
Tabelle 10: Übersicht über die verwendeten Simulationstools 117
Tabelle 11: Aufwände zur Einbindung der einzelnen Simulationstools 123
Tabelle 12: Vergleich des Dynamischen Co-Simulationskonzepts mit FMI 124

Abkürzungsverzeichnis

ACL	**A**gent **C**ommunication **L**anguage
API	**A**pplication **P**rogramming **I**nterface
BDI	**B**elief, **D**esire and **I**ntention
CPS	**C**yber-**P**hysisches **S**ystem
DDS	**D**ata **D**istribution **S**ervice
DES	**D**iscrete **E**vent **S**imulation
FMI	**F**unctional **M**ock-up **I**nterface
FMU	**F**unctional **M**ock-up **U**nits
HiL	**H**ardware-**i**n-the-**L**oop
ID	**Id**entifikator
IoT	**I**nternet **o**f **T**hings
JADE	**J**ava **A**gent **De**velopment Framework
HLA	**H**igh **L**evel **A**rchitecture
NFC	**N**ear **F**ield **C**ommunication
OPC UA	**O**pen **P**latform **C**ommunications **U**nified **A**rchitecture
OSGi	**O**pen **S**ervices **G**ateway **i**nitiative
RFID	**R**adio-**F**requency **Id**entification
RTI	**R**un-**T**ime-**I**nfrastructure
SOA	**S**ervice-**O**riented **A**rchitecture
STEP	**St**andard for the **E**xchange of **P**roduct model data
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WSN	**W**ireless **S**ensor **N**etworks

Zusammenfassung

Moderne automatisierte Systeme werden, bedingt durch unterschiedlichste Faktoren, zunehmend komplexer: So vernetzen sich Systeme durch das Internet der Dinge über Domänengrenzen hinweg, zudem werden sie deutlich modularer und flexibler gehalten und können sich zur Laufzeit verändern um sich an neue Gegebenheiten anzupassen und durch Ansätze wie „Plug-and-Play“ können neue, zur Entwurfszeit unbekannte Komponenten zur Laufzeit in Systeme eintreten.

Durch diese erhöhte Komplexität wächst auch der Bedarf an Unterstützung durch Simulation während des gesamten Lebenszyklus eines Systems. Allerdings überträgt sich diese Komplexität auch auf die Simulation solcher Systeme. Die Vernetzung über Domänengrenzen hinweg erhöht die Heterogenität der Systeme und ebenso die Heterogenität in der Simulation, weswegen eine Simulation solcher Systeme in einem einzelnen Simulationstool nahezu unmöglich wird. Zudem können verschiedene Komponenten effizienter in spezialisierten Simulationstools modelliert werden und Experten verwenden die von ihnen präferierten Simulationstools. Daher wird für die Simulation komplexer, vernetzter Systeme eine Co-Simulation benötigt. Darüber hinaus muss in der Simulation auch die Dynamik, im Speziellen das Ein- und Austreten von Komponenten zur Laufzeit, berücksichtigt werden. Daher sollte die Co-Simulation ebenfalls zur Laufzeit möglichst einfach erweiterbar sein um Ansätze wie „Plug-and-Play“ abbilden zu können.

Dies führt zur Zielsetzung dieser Arbeit: Es soll eine zur Laufzeit erweiterbare Co-Simulation ermöglicht werden, in die Teilsimulationen, den realen Komponenten gleich, zur Laufzeit per „Plug-and-Simulate“ ein- bzw.-austreten können, um eine nahtlose Simulation während des gesamten Lebenszyklus eines Systems zu ermöglichen.

Das in dieser Arbeit vorgestellte Konzept unterteilt die Co-Simulation in einzelne Komponentensimulationen, simuliert in ihren nativen Tools, die durch Vertreter gekapselt in die Co-Simulation zur Laufzeit ein- und austreten können. Um ein Eintreten per „Plug-and-Simulate“ zu ermöglichen, werden die Interaktionen zwischen den einzelnen Komponentensimulationen ebenfalls in sogenannten Interaktionssimulationen gekapselt, wodurch keine manuelle Kopplung der Komponentensimulation durchgeführt werden muss. Zusätzlich wird ein Schnittstellenkonzept zur aufwandsarmen Anbindung der Simulationstools vorgestellt, sowie eine Realisierungsmöglichkeit durch ein Co-Simulationsframework. Das Konzept der dynamischen Co-Simulation und der Anbindung der Simulationstools wurde dabei iterativ nach der Methode der Design Science und im Rahmen eines Industrieprojekts erstellt.

In dieser Kooperation konnte ein Transfer in die Industrie in Form eines Co-Simulationsframeworks geschaffen und ein Prototyp mit einem Technology Readiness Level (TRL) von 4 implementiert werden, anhand dessen das präsentierte Konzept evaluiert wurde.

Abstract

Modern automated systems are becoming increasingly complex due to a wide variety of factors: for example, systems are becoming networked across domain boundaries through the Internet of Things. They are also becoming much more modular and flexible and can change at runtime to adapt to new conditions, and approaches such as "plug-and-play" allow new components that were unknown at design time to enter systems at runtime.

This increased complexity also increases the need for simulation support throughout the lifecycle of a system. However, this complexity also transfers to the simulation of such systems. Networking across domain boundaries increases the heterogeneity of the systems and likewise the heterogeneity in the simulation, which is why a simulation of such systems in a single simulation tool becomes almost impossible. Moreover, different components can be modeled more efficiently in specialized simulation tools and experts use their preferred simulation tools. Therefore, co-simulation is needed for the simulation of complex, networked systems. In addition, the simulation must also take into account the dynamics, specifically the entry and exit of components at runtime. Therefore, the co-simulation should also be as easily extendable as possible at runtime in order to be able to map approaches such as "plug-and-play".

These aspects of heterogeneity and dynamics must therefore also be considered in the simulation, which leads to the objective of this thesis: A dynamic co-simulation has to be provided, in which sub-simulations can enter or exit at runtime, similar to the real components via "Plug-and-Simulate", in order to enable a seamless simulation during the entire life cycle of a plant.

The concept presented in this thesis divides the co-simulation into individual component simulations, simulated in different simulation tools, which can enter and exit the co-simulation at runtime and are encapsulated by simulation representatives. In order to enable an entry via "Plug-and-Simulate", the interactions between the individual component simulations are also encapsulated in so-called interaction simulations, so that no manual coupling of the component simulation has to be performed. In addition, an interface concept to integrate simulation tools is provided. The concept of the dynamic co-simulation and the integration of the simulation tools was iteratively developed according to the method of Design Science as part of an industrial project.

In this cooperation, a transfer to industry in form of a co-simulation framework was enabled and a prototype with a Technology Readiness Level (TRL) of 4 was implemented, to evaluate the presented concept.

1. Einleitung

In diesem einleitenden Kapitel wird zunächst die Motivation dieser Arbeit basierend auf aktuellen Trends dargelegt, aus welchen eine zentrale Hypothese für diese Arbeit formuliert wird. Anschließend werden die mit dieser Hypothese einhergehenden Herausforderungen beschrieben sowie die Zielsetzung dieser Arbeit definiert. Abschließend wird der restliche Aufbau der Arbeit skizziert.

1.1 Bedeutung von Simulation

Ein Trend der letzten Jahre im Bereich der Produktion ist die Entwicklung hin zu personalisierten und individualisierten Produkten [1]. Durch die Produktindividualisierung gibt es eine bedeutend höhere Varianz an Produkten, was wesentlich höhere Anforderungen in Bezug auf die Flexibilität von Produktionsanlagen mit sich bringt. So können Produktionslinien nicht mehr starr auf genau eine Produktvariante ausgerichtet sein, wie es zu Beginn der Massenproduktion der Fall war, sondern die Produktionsanlagen müssen heute wesentlich flexibler ausgelegt werden. Zudem werden die Lebenszyklen von Produkten durch steigende Innovationsraten immer kürzer, während die Lebensdauern der Produktionsanlagen gleichbleiben [2]. Dies erfordert einen regelmäßigen Umbau von Anlagen, was ebenfalls eine höhere Flexibilität der Anlagen voraussetzt. Somit wird eine erhöhte Flexibilität der Anlage sowohl in Hinblick auf die Fertigung einer höheren Variantenvielfalt, als auch in Hinblick auf eine leichtere Rekonfigurierbarkeit gefordert.

Der modulare Aufbau von Produktionsanlagen ist ein Ansatz, um die Flexibilität und vor allem die Rekonfigurierbarkeit, der Systeme zu erhöhen [3], [4]. Hierzu wird in [5] eine Produktionsanlage in einzelne Komponenten aufgeteilt, welche in ein Prozessleitsystem integriert werden. In diesem Fall existiert jedoch immer noch eine zentrale Steuerung der Anlage. Eine wesentlich höhere Flexibilität kann mit einer verteilten Steuerung erreicht werden, in der jede Komponente ihre eigene Steuerung mit sich bringt [6]. Wenn jede Komponente über ihre eigene Steuerung verfügt, können die einzelnen Komponenten untereinander ausgetauscht bzw. komplett ersetzt werden, ohne dass eine zentrale Steuerung des Produktionssystems angepasst werden muss. Somit verringert sich der Aufwand bei einer Rekonfiguration erheblich. Allerdings ist es möglich, dass einzelne Komponenten auf die neuen Bedingungen angepasst werden müssen. Der Einsatz von „Plug-and-Produce“ [7] ist ein weiterer Schritt, den Rekonfigurationsaufwand von Produktionssystemen zu verringern. Hierbei können einzelne Komponenten in einem Produktionssystem umgestellt werden bzw. neue Komponenten in dieses eintreten, ohne dass das Produktionssystem oder die betroffenen Komponenten angepasst werden müssen. Dadurch ist es ohne zusätzlichen Aufwand möglich, eine Produktionslinie allein durch ein neues Anordnen der Komponenten in die Lage zu versetzten, neue Produkte oder Produktvarianten zu produzieren. Der letzte, heute absehbare Schritt in dieser Entwicklung ist der Einsatz von Cyber-Physischen

Systemen und Komponenten, mit denen ein sehr hohes Maß an Flexibilität erreicht werden kann [8]. Cyber-Physische Systeme vereinen die physikalischen Eigenschaften der Systeme mit Datenverarbeitung und Kommunikationsfähigkeiten [9]. Durch diese Vereinigung ergeben sich wesentlich autonomere Systeme [10], die zudem dezentral organisiert sind und sich zur Laufzeit durch „Plug-and-Produce“ neu anordnen lassen, wodurch die Flexibilität von Produktionsanlagen noch zusätzlich gesteigert wird. Somit werden zukünftige Produktionssysteme modular und dezentral organisiert sein [11]. Da einzelne Komponenten mittels „Plug-and-Produce“ in das Produktionssystem ein und austreten, um die geforderte, höhere Flexibilität zu erreichen, werden diese Produktionssysteme wesentlich dynamischer. Auch wird es zunehmend öfter vorkommen, dass die Komponenten, die per „Plug-and-Produce“ in ein System eintreten zur Entwurfszeit des Systems nicht bekannt waren, sondern über eine Integration dieser Komponenten erst zur Betriebszeit entschieden wird.

Die Bestrebungen hin zu derartigen zukünftigen Produktionssystemen werden in Deutschland unter dem Begriff „Industrie 4.0“ zusammengefasst [12]. Allerdings lassen sich die Prinzipien der Cyber-Physischen Systeme auch auf andere Domänen übertragen, um ähnliche oder andere Vorteile zu erzielen. Medizinische Cyber-Physische Systeme zum Beispiel erleichtern die Überwachung und Behandlung von Patienten [13], in der Energieversorgung können Cyber-Physische Systeme bei der Überwachung und Steuerung der Energienetze eingesetzt werden [14] und im Personenverkehr können sie autonomes Fahren ermöglichen [15]. Durch die Kommunikationsfähigkeit der Cyber-Physischen Systeme werden in allen Lebensbereichen Komponenten vernetzt.

Diese Vernetzung in allen Bereichen des Lebens wird unter dem Begriff „Internet der Dinge (IoT)“ zusammengefasst [16]. [17] unterteilt das Internet der Dinge in neun Anwendungsbereiche, in welchen es auf unterschiedliche Weisen genutzt wird, allerdings immer mit dem Ziel, das Leben des Menschen zu erleichtern. Alle Bereiche haben gemein, dass dezentrale Systeme entstehen, in die zur Laufzeit neue Komponenten eintreten. Mit zunehmender Vernetzung kommunizieren somit immer mehr unterschiedliche Komponenten miteinander [18], sowohl Komponenten von unterschiedlichen Herstellern als auch über Domänengrenzen hinweg, wodurch die Systeme immer heterogener werden.

Die Dezentralisierung sowie die erhöhte Dynamik und Heterogenität der Systeme bringen unterschiedliche, neue Herausforderungen mit sich, wie beispielsweise in den Bereichen Simulation, Einhaltung von Qualitätsanforderungen wie Safety und Security, Semantik oder Standardisierung [19], [20].

Ein weiterer Trend der letzten Jahrzehnte ist, dass Simulation in allen Lebensphasen eines Produkts bzw. eines Systems eingesetzt wird [21]. In der Projektakquise ermöglicht sie schnell kompetente und detaillierte Antworten an Kunden zu geben, in der Entwurfsphase wird sie zur Unterstützung von Entwurfsentscheidungen eingesetzt [22], beim Testen können durch den

Einsatz von Hardware-in-the-Loop-Simulation Kosten gespart werden [23], die Inbetriebnahme kann durch Virtuelle Inbetriebnahme vereinfacht werden [24] und während des Betriebs kann Simulation sowohl zum Training von Personal [25] als auch zur Entscheidungsunterstützung bei der Prozessoptimierung genutzt werden [26]. Diese Verfügbarkeit einer Simulation während des gesamten Lebenszyklus wird unter dem Begriff Digitaler Zwilling zusammengefasst [27].

Da Systeme durch die Dezentralisierung immer komplexer werden [28], wird auch der Bedarf an Unterstützung durch Simulation während des gesamten Lebenszyklus eines Systems größer. Da IoT-Systeme dynamische Systeme sind, in die Komponenten jederzeit ein- und austreten können, ändern sich auch das Modell, bzw. die Einzelmodelle dieser Systeme ständig, vor allem, wenn Simulation während des Betriebs zur Entscheidungs-, Steuerungs-, Wartungs- und Überwachungsunterstützung eingesetzt wird. Somit wird durch die hohe Flexibilität des Systems eine ebenso hohe Flexibilität des Modells bzw. der Einzelmodelle gefordert und damit auch eine hohe Dynamik der Simulation. Die Herausforderung der Heterogenität überträgt sich ebenfalls auf die Simulation. Verschiedene Komponenten können effizienter in spezialisierten Simulationstools modelliert werden und Experten verwenden die von ihnen präferierten Simulationstools [29]. Somit wird es immer schwieriger ein komplexes heterogenes IoT-System in einer einzigen Simulation abzubilden, insbesondere, wenn unterschiedliche Aspekte des Systems, wie Kommunikation, physikalische Interaktionen und exaktes zeitliches Verhalten gleichzeitig simuliert werden sollen. Daher stellt sich die Frage, ob es überhaupt noch sinnvoll ist, alles in einer einzigen Simulation abzubilden. Wenn beispielsweise physikalische Vorgänge wie Temperaturverläufe oder Spannungen in Materialien simuliert werden müssen, ist es von Vorteil spezialisierte Solver zu verwenden [30]. Somit wird zur Simulation eines heterogenen IoT-Systems auch ein heterogenes Modell zum Einsatz kommen müssen. Ein weiterer Vorteil der Verwendung von spezialisierten Modellierungstools und -konzepten ist die Möglichkeit, sich auf bestimmte Aspekte zu fokussieren. Soll beispielsweise der Energieverbrauch eines Systems durch eine Simulation bestimmt werden, können Simulationstools zur Modellierung der Komponenten zum Einsatz kommen, welche eine detaillierte Modellierung des Energieverbrauchs ermöglichen. Soll hingegen die Abnutzung eines Systems untersucht werden, können für die Modellierung der Komponenten andere spezialisierte Simulationstools verwendet werden.

Somit lassen sich aktuell drei Haupttrends erkennen, die einen bedeutenden Einfluss auf die Simulation von modernen automatisierten Systemen haben, die ersten beiden Trends beziehen sich auf die simulierten Systeme und der dritte Trend beeinflusst die Simulationen selbst:

Trend 1: Systeme werden durch eine immer stärker werdende Vernetzung immer heterogener.

Trend 2: Systeme werden durch Bestrebungen der Rekonfigurierbarkeit mit dem Ziel von „Plug-and-Produce“ immer dynamischer und autonomer.

Trend 3: Simulationen und deren Modelle werden zunehmend während des gesamten Lebenszyklus eingesetzt.

Aus diesen Trends lässt sich folgende Hypothese formulieren:

Aktuelle Simulationskonzepte werden nicht sowohl der Heterogenität als auch der Dynamik moderner IoT-Systeme während des gesamten Lebenszyklus gerecht.

Diese Hypothese soll mit repräsentativen Beispiel belegt werden. Da, wie oben beschrieben, eine Vernetzung über Domänengrenzen hinweg stattfindet, wird ein Simulationskonzept dann der Heterogenität gerecht, wenn es möglich ist Komponenten aus unterschiedlichen Domänen zu simulieren. Und da, wie oben beschrieben, IoT-Komponenten zur Laufzeit in ein System ein- und austreten, wird ein Simulationskonzept der Dynamik dann gerecht, wenn dieses Ein- und Austreten von Komponenten ebenfalls simuliert werden kann.

Um der erhöhten Dynamik und Heterogenität bei der Simulation von IoT-Systemen gerecht zu werden, müssen die sich daraus ergebende Herausforderungen eindeutig identifiziert werden, um eine möglichst einfache und aufwandsarme Simulation von IoT-Systemen während des gesamten Lebenszyklus zu ermöglichen.

1.2 Herausforderungen der Simulation von IoT-Systemen

Solch ein neuartiges Simulationskonzept hat vier Herausforderungen für die Simulation von modernen automatisierten Systemen und IoT-Systemen im Allgemeinen zur Folge. Diese Herausforderungen finden sich immer wieder in der Literatur, beispielsweise [31], [32], [33], [34], [35]. In [31] wurde eine Befragung von über 200 Experten aus Forschung und Industrie zum Thema der Zukunft der Simulation durchgeführt, in der die nachfolgenden Herausforderungen eindeutig benannt werden, in [32] wurde diese Befragung 5 Jahre später nochmals, wieder mit über 200 Experten aus Forschung und Industrie, wiederholt, in welcher die Ergebnisse der ersten Befragung bestätigt wurden. Darüber hinaus wurden im Rahmen dieser Arbeit Experten auf dem Gebiet der Simulation aus Forschung und Industrie befragt, die diese Herausforderungen bestätigten. Zum einen fanden intensive Diskussionen mit mehreren Experten eines Industriepartners im Rahmen eines Industrieprojekts statt, und zum anderen wurden diese Herausforderungen von mehreren anderen Experten von namhaften Firmen in anderen Industrieprojekten und Diskussionsrunden mehrfach bestätigt. In [33] wird ebenfalls nochmal die Notwendigkeit einer „Plug-and-Simulate“-fähigen Co-Simulation hervorgehoben. In ihrer Roadmap für Technologien von 2020 ([34]) hat die NASA sowohl die zunehmende Relevanz von Co-Simulationen verdeutlicht, als auch die Notwendigkeit der Anpassbarkeit der Simulation und deren Modelle zur Laufzeit über den gesamten Lebenszyklus hinweg. Auch die Europäische Kommission hebt die Relevanz von Simulation während des gesamten Lebenszyklus sowie die einfache Erweiterbarkeit von Systemsimulationen in ihrer Roadmap von 2020 hervor [35].

Herausforderung 1 (H1): Die Simulationen der einzelnen IoT-Komponenten sollten getrennt voneinander in die Gesamtsimulation integrierbar sein.

Die Bestrebungen hin zur Rekonfigurierbarkeit bringen eine höhere Modularisierung mit sich, um in „Plug-and-Produce“-Szenarien möglichst leicht einzelne Komponenten austauschen zu können. Es ist wünschenswert, diese einfache Austauschbarkeit ebenfalls auf die Simulationswelt zu übertragen, so dass im Falle einer Rekonfiguration nicht nur die realen Komponenten per „Plug-and-Produce“ ausgetauscht werden können, sondern auch deren Modelle und somit deren Simulationen einfach per „Plug-and-Simulate“ ausgetauscht werden können. Somit ist es das Ziel, keine komplexe ineinander verwobene Einzelsimulation zu haben, sondern ebenfalls eine modular, komponentenorientiert aufgebaute Simulation [31], [32].

Herausforderung 2 (H2): Modelle und deren Simulationen sollten unabhängig voneinander agieren können.

Bestrebungen hin zu Autonomie sind ein weiterer Aspekt der Rekonfigurierbarkeit, mit dem Ziel, dass die einzelnen Komponenten eines Systems unabhängig voneinander autonom Aufgaben erfüllen können, ohne bei Eintritt einer derartigen Komponente manuelle Einstellungen und Anpassungen vornehmen zu müssen (Plug-and-Produce). Dies ist ähnlich zu Herausforderung 1, unterscheidet sich aber darin, dass in H1 die physische Modularität betrachtet wird, hier jedoch die funktionale Unabhängigkeit gemeint ist: Interaktionen zwischen Komponenten sollten ohne vorherigen menschlichen Eingriff erfolgen. Somit ist auch in der Simulation das Ziel diese funktionale Unabhängigkeit zu erreichen, so dass bei Eintritt einer neuen Simulation diese nahtlos mit anderen Simulationen interagieren kann und somit manuelle Eingriffe unnötig sind [31], [32]. Zudem sollten die simulierten Komponenten in ihren Tools auch unabhängig voneinander ausgeführt werden können, falls dies sinnvoll ist. Heißt, wenn eine IoT-Komponente auch ohne die Verbindung zu anderen IoT-Komponenten Aufgaben erfüllen kann, so muss deren Simulation dies auch ohne die Verbindung zu anderen Simulationen simulieren können.

Herausforderung 3 (H3): Es sollen verschiedene Simulationstools aus unterschiedlichen Domänen verwendet werden können.

Durch die immer höher werdende Vernetzung, auch über Domänengrenzen hinweg, werden IoT-Systeme immer heterogener. Diese Heterogenität wirkt sich auch auf die Simulationswelt aus: für die Simulation der einzelnen Komponenten werden verschiedene Tools aus unterschiedlichen Domänen verwendet [31], [32]. Die Gründe hierfür sind einerseits, da Simulationstools spezialisiert sind, ist es oft nicht möglich in einer sehr heterogenen IoT-Simulation alle Komponenten in einem einzigen Simulationstool zu simulieren. Andererseits werden Simulationen zunehmend während des gesamten Lebenszyklus hinweg eingesetzt, weshalb es wegen der Wiederverwendung sinnvoll ist, die bereits vom Komponentenhersteller erstellten Modelle zu verwenden. Allerdings benutzen unterschiedliche Hersteller unterschiedliche Simulationstools, was zu einer Verwendung von verschiedenen Simulationstools führt.

Herausforderung 4 (H4): Simulationen und deren Modelle sollen zur Laufzeit eintreten können.

Simulationen werden zunehmend während des gesamten Lebenszyklus eingesetzt und somit auch während des Betriebs eines Systems [31], [32]. Es ist daher von Vorteil, wenn, bedingt durch einen Eintritt einer neuen Komponente in das reale IoT-System zur Laufzeit, auch eine Simulation dieser neuen Komponente zur Laufzeit in die Simulation eintreten kann. Besonders während des Betriebs ist dies erstrebenswert, da hier bei IoT-Systemen, welche sehr dynamische Systeme sind, in die ständig neue Komponenten eintreten, beispielsweise in Form von neuen Produkten, oft die Zeit fehlt die gesamte Simulation zu stoppen und die neuen Teilsimulationen (auch Teilsimulationen, die zur Entwurfszeit der Simulation noch nicht bekannt waren) hinzuzufügen. Diese Herausforderung unterscheidet sich dadurch von Herausforderung H2, dass H2 eine funktionale Trennung der einzelnen Simulationen fordert, was, wie oben erläutert, dem Gedanken dezentraler IoT-Systeme entspricht. H4 fordert zusätzlich, dass diese Modelle mit einem minimalen manuellen Aufwand in die Gesamtsimulation integriert werden können. Minimaler manuelle Aufwand bedeutet, dass keine Anpassungen oder Änderung an der Gesamtsimulation, einer Teilsimulation, deren Modellen oder an den Verbindungen zwischen den Teilsimulationen vorgenommen werden müssen. Ausschließlich die Auswahl des Modells sowie das Aktivieren der Integration sollen manuell getätigt werden dürfen.

Während der frühen Lebenszyklusphasen ist es von Vorteil, Teilsimulationen zur Laufzeit in die Gesamtsimulation zu integrieren, da somit das Verhalten des Gesamtsystems auf Eintritte hin untersucht werden kann. Dasselbe gilt auch für Austritte aus dem Gesamtsystem bzw. aus der Gesamtsimulation.

Aus diesen Herausforderungen ergibt sich die zentrale Forschungsfrage für diese Arbeit:

Wie können Teilsimulationen mit minimalem manuellen Aufwand zur Laufzeit in eine Gesamtsimulation eines typischen IoT-Systems integriert werden?

Minimaler manueller Aufwand bedeutet, dass beim Integrieren neuer Teilsimulationen keine Anpassungen oder Änderung an der Gesamtsimulation, einer Teilsimulation, deren Modellen oder an den Verbindungen zwischen den Teilsimulationen vorgenommen werden müssen. Ausschließlich die Auswahl des Modells sowie das Aktivieren der Integration sollen manuell getätigt werden dürfen.

1.3 Zielsetzung der Arbeit

Aus der im vorherigen Unterkapitel definierten Forschungsfrage ergibt sich folgende Zielsetzung dieser Arbeit:

Entwicklung eines Konzepts, welches es ermöglicht Teilsimulationen in eine Gesamtsimulation eines IoT-Systems zur Laufzeit mit minimalem manuellen Aufwand zu integrieren.

Die im vorherigen Unterkapitel hergeleiteten Herausforderungen bilden den Ausgangspunkt für die vorliegende Arbeit. Um diese Zielsetzung zu erfüllen, müssen durch solch ein Konzept Mehrwerte erfüllt werden, welche im Folgenden genauer erörtert werden.

Es soll möglich sein, mit nur minimalem manuellen Aufwand neue Teilsimulationen in eine Gesamtsimulation zur Laufzeit einzubinden, ohne diese stoppen zu müssen, was den ersten Mehrwert M1 darstellt. Minimaler manueller Aufwand bedeutet, dass beim Integrieren neuer Teilsimulationen keine Anpassungen oder Änderung an der Gesamtsimulation, einer Teilsimulation, deren Modellen oder an den Verbindungen zwischen den Teilsimulationen vorgenommen werden müssen. Ausschließlich die Auswahl des Modells sowie das Aktivieren der Integration sollen manuell getätigt werden dürfen. Dieser Mehrwert ergibt sich aus den Herausforderungen H1, H2 und H4. Herausforderung 1 fordert, dass Simulationen unabhängig von anderen Simulationen in die Gesamtsimulation eingebunden werden können sollten. Somit darf hierbei nur ein minimaler manueller Aufwand zur Anpassung der anderen Simulationen entstehen. Herausforderung 2 fordert, dass Interaktionen zwischen Komponenten ohne vorherigen menschlichen Eingriff erfolgen sollen, um eine nahtlose Integration neuer Teilsimulationen zu ermöglichen. Somit darf beim Einbinden kein manueller Aufwand an dieser oder an den anderen Simulationen entstehen, um eine Interaktion zwischen den Simulationen zu ermöglichen. Somit darf hierbei kein manueller Aufwand zur Anpassung oder Änderung an der einzubindenden Simulation, anderen Teilsimulationen oder an deren Verbindungen notwendig sein. Daher darf insgesamt kein manueller Aufwand beim Einbinden von neuen Simulationen in die Gesamtsimulation entstehen, außer der Auswahl des Modells sowie das Aktivieren der Integration. Herausforderung 4 fordert, dass das Einbinden von Simulationen zur Laufzeit möglich sein soll, ohne die anderen Simulationen pausieren oder stoppen zu müssen.

Herausforderung 3 fordert, dass es möglich sein muss, unterschiedliche Simulationen aus unterschiedlichen Domänen einzusetzen. Hieraus leitet sich direkt der zweite Mehrwert (M2) ab, nämlich, dass es möglich sein soll, zahlreiche unterschiedliche Simulationstools, auch aus verschiedenen Domänen, verwenden zu können, um die Teilsimulationen auszuführen.

In Herausforderung 4 wird zudem gefordert, dass auch zur Entwurfszeit des Systems unbekannt Komponenten eintreten können müssen. Dies soll auf die Simulationswelt übertragen werden, was sich in dieser zusätzlich zu neuen, unbekannten, simulierten Komponenten, darin auswirkt, dass auch neue, unbekannte Simulationstools integriert werden müssen. Dies ist beispielsweise der Fall, wenn eine neue Komponente eines neuen Zulieferers eingebunden werden soll und dieser ein eigenes Simulationstool für die Simulation dieser Komponente verwendet. Herausforderung H3 fordert zudem, dass genau solche vom Zulieferer bereitgestellten Simulationen ebenfalls in der Gesamtsimulation verwendet werden sollen. Hieraus leitet sich nun der dritte Mehrwert (M3) ab, dass es möglich sein soll, neue Simulationstools mit einem möglichst geringen Aufwand an die Gesamtsimulation anzubinden.

Um diese Mehrwerte bieten zu können und um die genannten Herausforderungen erfüllen zu können, muss ein Konzept die folgenden Anforderungen erfüllen. Hierbei dienen die folgenden Anforderungen lediglich dazu, die Zielstellung dieser Arbeit zu präzisieren und dazu, die Erfüllung der Herausforderungen und der daraus resultierenden Zielstellung, nachprüfen zu können.

Anforderung 1 (A1): Um Herausforderung H1 zu erfüllen, müssen Modelle und deren Simulationen hinzugefügt oder entfernt werden können, ohne dass das Gesamtmodell manuell angepasst werden muss. Es darf bei der Integration neuer Teilmodelle kein Modellierungsaufwand an diesem oder an anderen Modellen nötig sein. Hiermit ist ein Modellierungsaufwand an den Modellen selbst, nicht an den Schnittstellen der Modelle gemeint.

Anforderung 2 (A2): Um Herausforderung H2 zu erfüllen, müssen Modelle und deren Simulationen hinzugefügt oder entfernt werden können, ohne dass die möglichen Interaktionen mit anderen Modellen und deren Simulationen manuell angepasst werden müssen. Es darf bei der Integration neuer Teilmodelle kein Modellierungsaufwand an den Interaktionen der einzelnen Modelle nötig sein.

Anforderung 3 (A3): Um Herausforderung H3 zu erfüllen, müssen die Teilmodelle in unterschiedlichen Simulationstools aus unterschiedlichen Domänen simulierbar sein. Somit müssen sowohl unterschiedlichste Simulationstools integrierbar sein als auch der Aufwand der Integration eines neuen Simulationstools muss möglichst gering sein.

Anforderung 4 (A4): Um Herausforderung H4 zu erfüllen, muss die Gesamtsimulation zur Laufzeit um weitere Teilsimulationen erweiterbar sein. Zur Integration von neuen Teilsimulationen, auch von zur Entwurfszeit der Gesamtsimulation unbekannten Teilsimulationen, darf kein Stoppen der Gesamtsimulation nötig sein.

Die Relevanz dieser Anforderungen wurden in mehreren intensive Diskussionen mit mehreren Experten eines Industriepartners im Rahmen eines Industrieprojekts und von mehreren anderen Experten von namhaften Firmen in anderen Industrieprojekten und Diskussionsrunden mehrfach bestätigt. Werden diese Anforderungen von dem zu erstellenden Konzept erfüllt, werden somit auch die in Kapitel 1.2 hergeleiteten Herausforderungen erfüllt.

1.4 Abgrenzung der Arbeit

Im Rahmen dieser Arbeit wird ein Konzept für eine „Plug-and-Simulate“-fähige Simulation von IoT-Systemen entwickelt. Somit liegt der Fokus dieser Arbeit auf der Simulation des Gesamtsystems und dem Einbinden von neuen Teilsimulationen zur Laufzeit, nicht auf den Teilsimulationen selbst. Ein Konzept zur Modellierung von einzelnen IoT-Komponenten sowie funktionierende Teilsimulationen werden aus anderen Arbeiten entliehen, welche diese Aspekte genauer betrachten. Diese Teilsimulationen müssen sowohl die Funktionsweise der von ihnen

simulierten Komponente enthalten als auch deren Fähigkeiten bezüglich Kommunikation und zur physikalischen Interaktion mit anderen Komponenten bzw. deren Modellen. Hierbei wird weiterhin vorausgesetzt, dass die Kommunikation und physikalischen Interaktionen in einem hinreichenden Detailgrad modelliert und somit simulierbar sind. Dieser hinreichende Detailgrad wird in Kapitel 3.3 genauer definiert.

1.5 Aufbau der Arbeit

Die vorliegende Arbeit ist in 7 Kapitel gegliedert, vgl. Abbildung 1. In Kapitel 2 wird zunächst die aktuelle Literatur zu Simulation im Allgemeinen untersucht, danach wird auf die aktuelle Literatur zum Internet der Dinge (IoT) eingegangen. Anschließend werden bestehende Ansätze zur Simulation von IoT-Szenarien beschrieben, gefolgt von bestehenden Ansätzen zur dynamischen Co-Simulation. Diese Ansätze werden hinsichtlich der in Kapitel 1 aufgestellten Anforderungen untersucht und abschließend wird eine Forschungslücke aufgezeigt.

Kapitel 3 gibt zuerst einen Überblick über das entwickelte Gesamtkonzept. Daraufhin werden die einzelnen Teile des Gesamtkonzepts detaillierter beschrieben, zuerst die Umsetzung des Datenaustauschs in der Co-Simulation, dann das Konzept der Interaktionssimulationen und abschließend die Durchführung der Synchronisation der gesamten Co-Simulation. Im letzten Unterkapitel von Kapitel 3 wird beschrieben, wie reale Komponenten in das vorgestellte Konzept integriert werden können.

In Kapitel 4 werden realisierungstechnische Aspekte und Entscheidungen dargelegt. Dies beinhaltet die Umsetzung der Co-Simulationsumgebung und der Simulationsvertreter, ein Konzept zur Umsetzung der Schnittstellen sowie ein Konzept für eine Modellbibliothek der Interaktionssimulationen.

In Kapitel 5 folgt eine Beschreibung der Implementierung dieser Aspekte und Entscheidungen.

Die Evaluierung wird in Kapitel 6 beschrieben, aufgegliedert in die Beschreibung des Evaluierungsszenarios, die Erfüllung der Herausforderungen und die Erfüllung der Zielsetzung.

In Kapitel 7 werden die wichtigsten Aspekte dieser Arbeit zusammengefasst und die erzielten Ergebnisse aufgeführt. Abschließend werden Möglichkeiten zur Fortführung der Arbeit aufgezeigt.

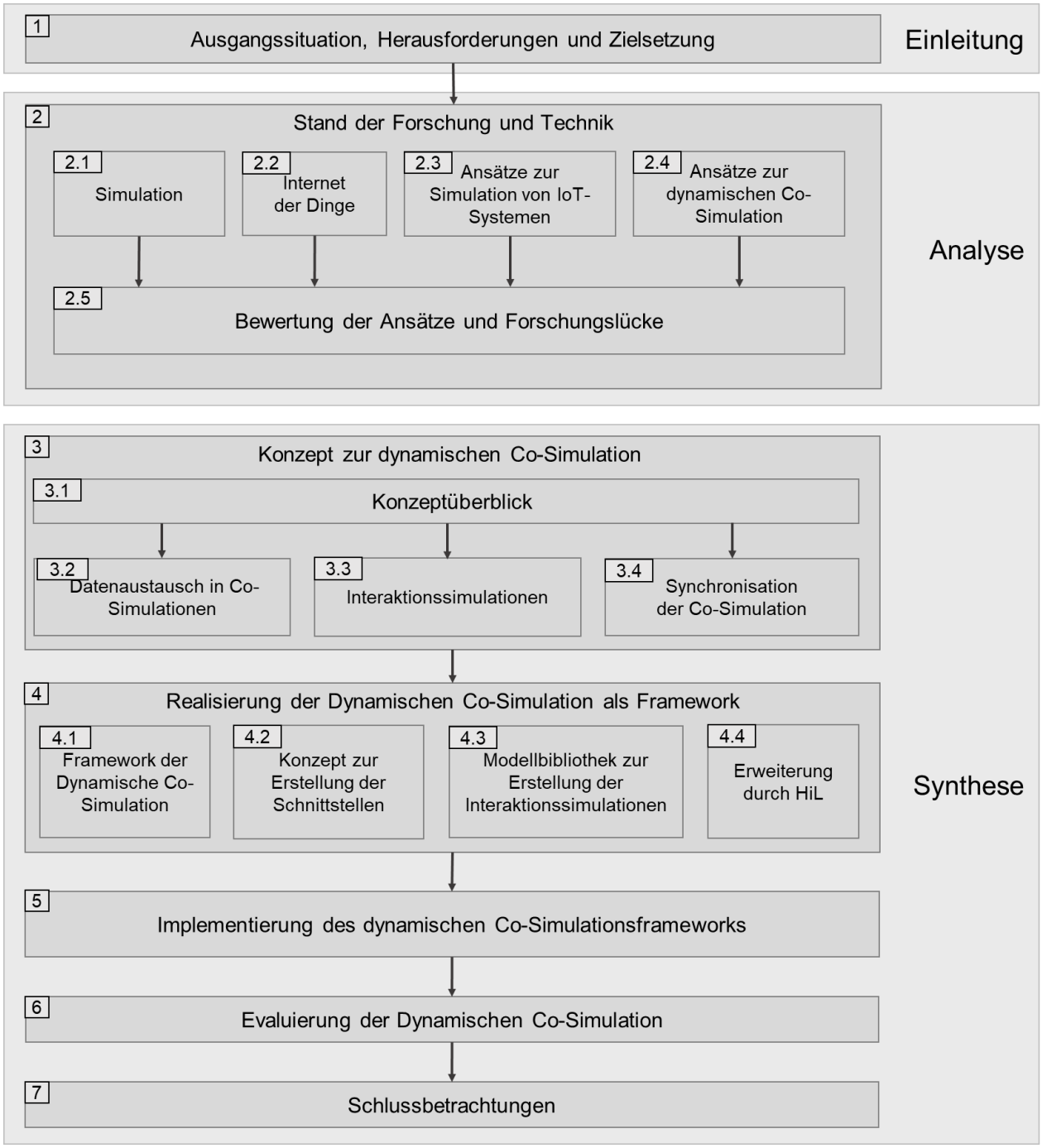


Abbildung 1: Aufbau der Arbeit

2 Stand der Forschung und Technik

In diesem Kapitel werden zuerst aktuelle Entwicklungen im Bereich der Simulation (Kapitel 2.1) und anschließend das Internet der Dinge (Kapitel 2.2) beleuchtet. Im Bereich der Simulation werden zunächst aktuelle Entwicklungen der Simulation beschrieben und anschließend grundlegende Simulationskonzepte erläutert, welche für die anschließende Diskussion des Stands der Technik und Forschung benötigt werden und für die spätere Modellierung der Gesamtsimulation relevant sind. Zum Internet der Dinge werden aktuelle Entwicklungen aufgezeigt und ein Überblick gegeben, welche Aspekte im Internet der Dinge relevant sind, insbesondere welche Aspekte in einer Simulation eines Internet-der-Dinge-Szenarios berücksichtigt werden müssen.

Anschließend (Kapitel 2.3) werden aktuelle Veröffentlichungen zur Simulation von IoT-Szenarien vorgestellt. Hierbei handelt sich um einzelne Simulationen, welche einzelne Aspekte des IoT-Szenarios abbilden. Diese unterteilen sich in Diskrete-Event-Simulatoren und Agentenbasierte Simulatoren.

In dem folgenden Teilkapitel 2.4 werden bestehende Ansätze zur dynamischen Co-Simulation vorgestellt. Diese unterteilen sich in den Stand der Technik (Functional Mock-up Interface und High Level Architecture) und den Stand der Forschung (SOA, OSGi, Agenten sowie allgemeine Ansätze aus der Literatur). Da die Agenten für die spätere Realisierung relevant sind, wird bei der Diskussion der Co-Simulationsansätze mit Agenten auch kurz das Konzept der Agenten vorgestellt.

Zuletzt erfolgt eine abschließende Diskussion der vorgestellten Standards und Ansätze (Kapitel 2.4.9) aus welcher anschließend eine Forschungslücke abgeleitet wird.

2.1 Simulation

Simulation ist das Nachbilden eines dynamischen Prozesses in einem System mit Hilfe eines experimentierfähigen Modells, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind [36]. Hierbei ist ein Modell eine vereinfachte Nachbildung eines geplanten oder existierenden Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System [36].

Simulation wird aus unterschiedlichen Gründen eingesetzt. Gründe für die Simulation einer Anlage können zum Beispiel die Unterstützung während der Entwicklung und des Testens, die Optimierung von Prozessen während des Betriebs oder das Training von Personal sein [37]. Simulation wird immer dann eingesetzt, wenn ein Erkenntnisgewinn am realen System zu teuer, zu aufwändig oder aus anderen Gründen nicht sinnvoll oder möglich ist oder wenn eine analytische Lösung einer Problemstellung nicht möglich oder zu aufwendig ist [38]. Bei

Simulation kann zwischen physikalischen Modellen und mathematischen Modellen unterschieden werden. Physikalische Modelle sind physikalische Nachbauten eines Systems, wie sie beispielsweise in Windkanälen eingesetzt werden. Mathematische Modelle hingegen beschreiben ein System mittels logischer und quantitativer Zusammenhänge und die Reaktion des Systems auf bestimmt Ereignisse [39]. Die Simulation mit mathematischen Modellen wird als Computersimulation bezeichnet [40] und im Weiteren wird der Begriff Simulation für diese Art der Simulation synonym verwendet.

2.1.1 Bedeutung von Simulation

Simulation wurde erstmals in den 1960ern häufiger zur Unterstützung in der Entwicklung verwendet, allerdings nur von Experten für spezifische Probleme. Heute ist sie ein gängiges Tool für die Entwicklungsunterstützung und zum Testen von Systemen. Allerdings ist ihr Gebrauch hauptsächlich auf Forschungs- und Entwicklungsabteilungen begrenzt [41]. In [41] wird der Digitale Zwilling als nächste Stufe der Simulation vorausgesagt. Der Digitale Zwilling ist eine multiphysikalische, skalierbare Simulation eines Systems, die ein ultrarealistisches, ständig aktuelles Abbild dieses Systems darstellt. Der Digitale Zwilling erfasst zudem sämtliche Prozessdaten des Systems [42]. Mit dieser Vision wird in Zukunft ständig eine Simulation parallel zu einem System laufen. In Abbildung 2 sind diese Phasen in der Entwicklung der Simulation über die letzten Jahrzehnte hinweg dargestellt.

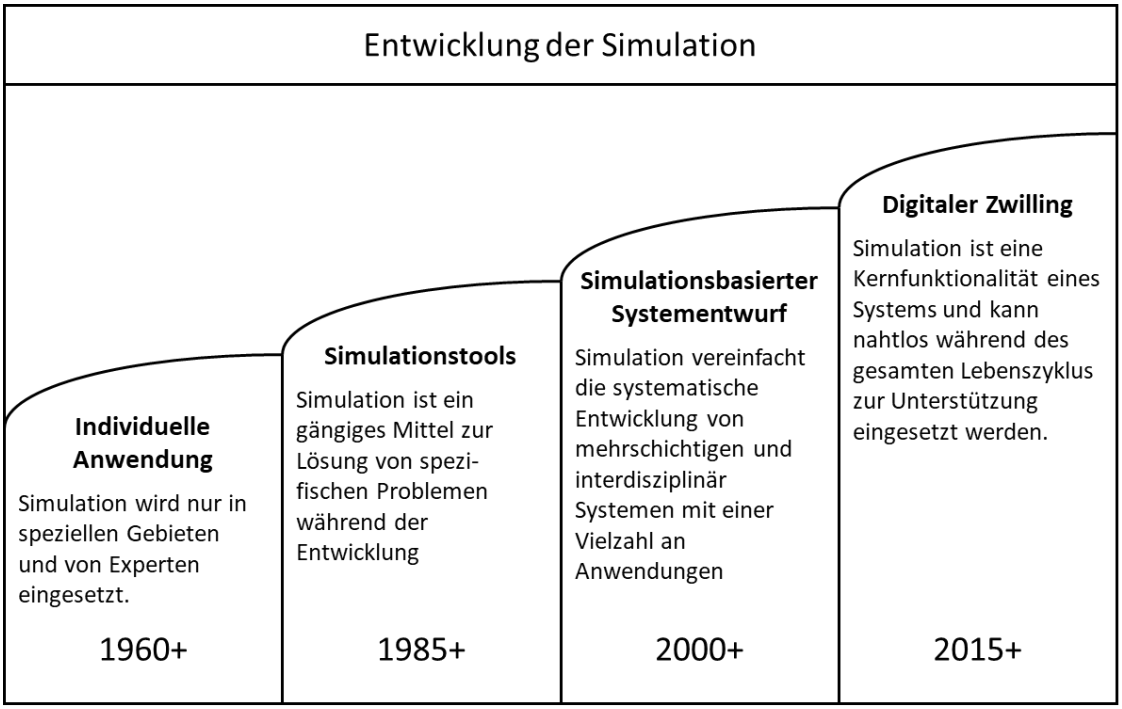


Abbildung 2: Entwicklung der Simulation über die Zeit [41]

In [43] wird ein deutlicher Anstieg der Bedeutung der Simulation in den nächsten Jahren vorausgesagt. Besonders durch eine nahtlose Simulation während des gesamten Lebenszyklus können zusätzliche Mehrwerte geschaffen werden [37]. In [31] wurde eine Umfrage durchgeführt,

die diese Steigerung der Bedeutung der Simulation in Zukunft bestätigt. Auch durch den Einsatz von mechatronischen Komponenten beziehungsweise Cyber-Physischen Systemen werden Systeme immer komplexer wodurch die Notwendigkeit des Einsatzes von Simulation während des gesamten Lebenszyklus ständig steigt [44].

Gerade bei Cyber-Physischen Systemen spielt die Vernetzung der Komponenten und Systeme untereinander eine große Rolle, wodurch sich auch hier Internet-der-Dinge-Systeme bilden. Somit stellt das IoT eine parallele Entwicklung zum Digitalen Zwilling dar. Wird nun dieser Anstieg der Bedeutung der Simulation mit den Entwicklungen von IoT-Systemen verknüpft, wird nochmals deutlich, dass ebenfalls neue Simulationskonzepte entwickelt werden müssen. Technologische Neuerungen welche durch das Internet der Dinge vorangetrieben werden, aber auch in Simulationen des Digitalen Zwillings berücksichtigt werden müssen, werden in Kapitel 2.2.1 vorgestellt.

Eine besondere Art der Simulation, stellt die Co-Simulation dar, bei der unterschiedliche Simulationen verknüpft werden, um eine Gesamtsimulation zu bilden. Durch den Digitalen Zwilling wird diese Art der Simulation besonders interessant, da es hierdurch möglich wird, die einzelnen Simulationen der einzelnen Digitalen Zwillinge miteinander zu verknüpfen und somit eine Simulation eines größeren Systems ermöglicht. Die Co-Simulation wird genauer in Kapitel 2.4.1 beschrieben.

2.1.2 Klassifikation von Simulationsarten

Die Simulation mit mathematischen Modellen lässt sich durch verschiedene Kategorien klassifizieren. Es gibt eine Vielzahl an möglichen Klassifizierungsversuchen, hier soll exemplarisch die Klassifizierung von A. Gupta [39] genommen werden, da dieser gut die große Vielfalt an möglichen Modellen darstellt. Es besteht allerdings kein Anspruch auf Vollständigkeit, sondern es soll lediglich ein erster Überblick gegeben werden.

Statische vs. Dynamische Modelle

Statische Modelle repräsentieren den Zustand eines Systems zu einem bestimmten Zeitpunkt oder werden für Systeme verwendet, die sich nicht über die Zeit ändern. Ein Beispiel für eine statische Simulation ist die Monte-Carlo-Simulation, mit der beispielsweise Probleme in der Nuklearphysik simuliert werden [45]. Dynamische Modelle hingegen werden für Systeme benötigt, die sich im Laufe der Zeit ändern, wie beispielsweise die Simulation von Temperaturverläufen.

Deterministische vs. Stochastische Modelle

Hängt ein System nicht vom Zufall ab, wird es als deterministisch bezeichnet. Hierfür werden Modelle verwendet, die bei gleichbleibender Ausgangssituation immer das gleiche Ergebnis liefern. Ein Beispiel hierfür ist die Simulation von elektrischen Schaltungen. Stochastische Modelle hingegen enthalten Zufallsvariablen und liefern somit nicht immer das gleiche Ergebnis,

selbst wenn die Ausgangssituationen identisch waren. Stochastische Simulationen werden oftmals bei Systemen eingesetzt, in die Objekte für das System zufällig über die Zeit verteilt eintreten, wie beispielsweise Warenlager.

Kontinuierliche vs. Diskrete Modelle

Kontinuierliche Modelle werden für Systeme verwendet, welche sich kontinuierlich über die Zeit ändern, wie beispielsweise bei der Simulation physikalischer Vorgänge, und können durch Differentialgleichungen abgebildet werden. Allerdings werden für solche Probleme die Modelle oftmals diskretisiert, da das Lösen der Differentialgleichungen zu aufwändig oder komplex ist [46]. Diskrete Modelle kommen auch bei Systemen zum Einsatz, die selbst diskret sind oder sehr einfach und sinnvoll diskretisiert werden können, wie beispielsweise Kommunikationsnetzwerke [47]. In Rechnern findet immer zwingend eine Diskretisierung statt, da in Rechnern immer diskrete Prozess ablaufen [48].

Je nach zu simulierendem System wird ein entsprechendes Modell benötigt, dem Modell entsprechend werden unterschiedliche Simulationsarten benötigt. Hier soll ein Überblick über verschiedene Simulationsarten gegeben werden, allerdings auch hier ohne Anspruch auf Vollstäbndigkeit:

Simulation mit Differentialgleichungen

Bei der Simulation mit Differentialgleichungen werden dynamische, deterministische, kontinuierliche Systeme durch Differentialgleichungen beschrieben. Beispiele hierfür sind physikalische Vorgänge, deren Differentialgleichungen durch einen Rechner lösbar sind wie der freie Fall eines Objekts [49].

Numerische Lösungsverfahren von Differentialgleichungen

Wird das Lösen der Differentialgleichungen zu aufwändig oder komplex, so können die Differentialgleichungen im Modell diskretisiert werden. Ein Beispiel für ein numerisches Lösungsverfahren ist die Finite-Elemente-Methode [46].

Monte-Carlo-Simulation

Mit einer Monte-Carlo-Simulation werden stochastische Systeme modelliert und simuliert. Dabei werden Lösungen von Problemen, welche nicht oder nur sehr schwer analytisch lösbar sind, durch statistische Verteilungen angenähert, wie zum Beispiel die Berechnung von π [50].

Systemdynamische Simulation

Bei der systemdynamischen Simulation werden Differentialgleichungen von dynamischen, kontinuierlichen, deterministischen Systemen durch Wirkungsketten und Flussdiagramme modelliert. Somit wird die Handhabung der Differentialgleichungen für den Nutzer vor allem bei komplexen Systemen vereinfacht [51].

Diskrete-Event-Simulation

Die Diskrete-Event-Simulation wird zur Simulation von Diskrete-Event-Systemen verwendet und wird in Kapitel 2.3.1.1 näher beschrieben.

Agentenbasierte Simulation

Bei der agentenbasierten Simulation werden die einzelnen Komponenten eines Systems durch Agenten modelliert, welche miteinander interagieren können. Sie wird in Kapitel 2.3.2.1 näher beschrieben.

2.1.3 Simulationsphasen

Klassischer Weise müssen zur Erstellung und Durchführung einer Simulation mehrere Phasen durchgeführt werden, welche oftmals mehrfach durchlaufen werden müssen. Auch hier gibt es unterschiedliche Phasenmodelle, beispielhaft soll hier das Phasenmodell nach [52] beschrieben werden, da dies einen guten Überblick über die benötigten Vorgänge gibt. Dort werden folgende Phasen einer Simulation definiert:

Modellierung: Als Erstes wird ein Modell des zu simulierenden Systems erstellt. Hierbei erfolgt eine Fokussierung auf die relevanten Aspekte des Systems, um eine möglichst ressourcenschonende Simulation zu ermöglichen.

Berechnung: Die im Modell enthaltenen Berechnungsalgorithmen müssen aufbereitet werden, damit es auf einem Computer verarbeitet werden kann und geeignete Algorithmen ausgewählt werden können.

Visualisierung: Eine weitere Herausforderung bei der Simulation ist die Aufbereitung und Darstellungen der Simulationsergebnisse, welche entsprechend visualisiert werden müssen.

Validierung: Die Simulationsergebnisse müssen noch zusätzlich validiert und überprüft werden, da im Modell selbst oder bei seiner Verarbeitung Fehler auftreten können, welche zu einer Abweichung des Simulationsergebnisses von einem realen Ergebnis führen können.

Einbettung: Oftmals muss die Simulation noch in einem entsprechenden Kontext eingebettet werden, wie zum Beispiel durch Integration in einen Entwicklungsprozess.

Werden die einzelnen Phasen mehrfach durchlaufen, so gehen die aus einem vorhergegangenen Durchlauf gewonnenen Erkenntnisse in den nächsten Durchlauf mit ein. Hierdurch hängen die einzelnen Phasen stark voneinander ab und können nicht komplett voneinander getrennt behandelt beziehungsweise umgesetzt werden. Unter Simulation werden klassischer Weise hauptsächlich die ersten beiden Phasen verstanden [52].

In dieser Arbeit müssen diese einzelnen Phasen ebenfalls berücksichtigt werden, allerdings werden in dieser Arbeit diese Phasen nicht selbst durchlaufen, stattdessen dient das Ergebnis

dieser Arbeit als Unterstützung dieser Phasen für den späteren Nutzer einer IoT-Simulation. Hierbei werden hauptsächlich die Phasen Modellierung, Berechnung und Visualisierung unterstützt, und zwar mit Hinweisen bei der Modellierung der einzelnen IoT-Komponenten, um eine Gesamtsimulation zu ermöglichen, mit der Bereitstellung einer Verknüpfung der einzelnen Teilsimulationen, um die Berechnung der Gesamtsimulation zu ermöglichen und mit der Bereitstellung einer Visualisierung der Gesamtsimulation, die die Zusammenhänge zwischen den einzelnen Teilsimulationen aufzeigt.

Da in dieser Arbeit die Simulation von Internet-der-Dinge-Systeme betrachtet wird, werden im folgenden Unterkapitel für die Simulation relevante Aspekte des IoTs untersucht. Zuerst wird untersucht, welche neuartigen Technologien (Enabling-Technologien) des IoT in einer Simulation berücksichtigt werden müssen und anschließend wird untersucht, welche Aspekte der elementaren Anwendungsfälle ebenfalls in einer Simulation berücksichtigt werden müssen.

2.2 Internet der Dinge

Das Prinzip des Internet der Dinge wurde erstmals von Mark Weiser 1991 in seinem Aufsatz „The Computer for the 21st Century“ vorgestellt [53]. Der Begriff „Internet of Things (IoT)“ selbst geht auf Kevin Ashton zurück, der ihn 1999 zum ersten Mal in einem Vortrag erwähnte [54]. Schon 1990 wurde ein Toaster über das Internet angesteuert und gilt heute offiziell als das erste Gerät im IoT [55]. In seinen Anfängen wurden im Internet der Dinge einzelnen Objekte durch RFID-Tags identifiziert [56]. Diese eindeutige Identifikation von Objekten ist auch heute noch ein Kernmerkmal des Internet der Dinge [57]. Heute sind schon mehrere Milliarden Geräte mit dem Internet verbunden und diese Zahl wird in den nächsten Jahren stark steigen [58]. Hierbei sind Geräte ohne Internetzugang, die aber trotzdem eine eindeutige Identifizierung über RFID oder ähnliches ermöglichen, nicht mit eingerechnet. Auf Grund dieser hohen Zahl an vernetzten und identifizierbaren Geräten werden Untersuchungen zur eindeutigen Identifikation von Geräten und Objekten angestellt. Diese beinhalten verschiedene Identifizierungsprotokolle wie IPv4 oder IPv6 [57], Electronic Product Code (EPC) [59] oder ucode [60].

Neben der eindeutigen Identifikation ist die Vernetzung der einzelnen Objekte essentiell für das IoT [61]. Somit lässt sich, das „Internet der Dinge“ wie folgt definieren:

„Das „Internet der Dinge“ ist eine dynamische, globale Infrastruktur mit der Fähigkeit zur Selbstkonfigurierung, welche auf standardisierten und interoperablen Kommunikationsprotokollen basiert, und in der physische und virtuelle „Dinge“ Identitäten, physikalische Attribute, virtuelle Persönlichkeiten besitzen, intelligente Schnittstellen benutzen und nahtlos in das Informationsnetzwerk integriert sind" [62].

2.2.1 Schlüsseltechnologien im IoT

In diesem Unterkapitel werden Schlüsseltechnologien des Internet der Dinge untersucht und anschließend erläutert welche Schlüsseltechnologien bzw. welche zugrundeliegenden Konzepte dieser Schlüsseltechnologien eine Relevanz für die Simulation eines IoT-Systems haben und deshalb in einer Simulation dieses berücksichtigt werden müssen.

Im IoT steht die Interaktion einzelner Objekte durch Kommunikation zum Erreichen gemeinsamer übergeordneter Ziele im Vordergrund [61]. Zum Erreichen dieser Interaktion zwischen den einzelnen Geräten werden mehrere Enabling-Technologien aus unterschiedlichen Bereichen benötigt [63]. Zusätzlich zu Identifizierungsprotokollen werden Technologien aus den Bereichen Kommunikationstechnologien, Kommunikationsprotokolle, Middleware, Betriebssysteme, Semantik, Security und Hardware benötigt [19], [63].

Im Bereich der Kommunikation werden ständig neue Übertragungstechnologien wie ZigBee, Z-Wave oder RuBee vorgestellt, welche speziell für die Kommunikation zwischen Geräten bzw. Sensoren entwickelt wurden [64]. Diese neuen Technologien zielen meistens auf einen geringeren Energieverbrauch ab, als herkömmliche Übertragungstechnologien wie z. B. WLAN. Auch Bluetooth Low Energy wurde im Gegensatz zu Bluetooth für einen geringeren Energieverbrauch entwickelt [65]. Zusätzlich wurden in den letzten Jahren neuartige Protokolle veröffentlicht, die für die Machine-to-Machine-Kommunikation entwickelt wurden, wie beispielsweise MQTT (Message Queuing Telemetry Transport), XMPP (Extensible Messaging and Presence Protocol) und Data Distribution Service (DDS). Diese Protokolle wurden für Netzwerke mit Restriktionen, beispielweise in der Bandbreite oder der Verfügbarkeit, entwickelt.

Auch in der Energieversorgung von Geräten und Sensoren besteht Entwicklungsbedarf, da viele Geräte und Sensoren längere Zeit im Feld sind und keinen Energienetzzugang besitzen. Energy-Harvesting ist hier eine Lösung die schon eingesetzt wird, bei der beispielsweise aus elektromagnetischen Wellen Energie gewonnen wird [66]. Zusätzlich können im Bereich der Identifikation passive Technologien wie RFID oder NFC eingesetzt werden, bei denen die benötigte Energie von einem Lesegerät bereitgestellt wird.

Ein weiterer treibender Technologiebereich sind Hardwarelösungen. Hierunter fallen beispielsweise RFID-Sensoren, welche in Baumaterialen [67] oder sogar den menschlichen Körper [68] implantiert werden können und ohne eigene Energiequelle, sondern nur über die durch das Lesegerät gewonnene Energie Sensorwerte erfassen und übermitteln können. Außerdem werden Single-Board Computer [69] immer leistungsfähiger, wodurch viele Geräte mit Rechenleistung und Kommunikationsfähigkeiten ausgerüstet werden können.

Auch in den Bereichen Semantik, Security und Privacy besteht noch hoher Forschungsbedarf, um das volle Potential des IoTs nutzen zu können [18], auch wenn in den letzten Jahren schon mehrere Beschreibungssprachen wie SensorML und SSN-Ontology für Sensornetzwerke entwickelt

wurden. Weitere offene Probleme verbleiben durch die enormen Skalierungen, zu denen es im IoT kommen kann. Zudem müssen Konzepte entwickelt werden, um aus den Daten, die im IoT anfallen, beispielsweise durch Sensornetzwerke, Nutzen ziehen zu können [70].

All diese neuartigen Technologien und Konzepte sind prinzipiell relevant für eine Simulation eines IoT-Szenarios. Der Fokus dieser Arbeit liegt hauptsächlich auf der Erstellung einer Gesamtsimulation und nicht auf der Modellierung der einzelnen IoT-Komponenten, daher sind für diese Arbeit hauptsächlich die Technologien und Konzepte relevant, deren Einfluss sich nicht auf die einzelnen Komponenten beschränkt. Diese sind Kommunikationstechnologien und -protokolle sowie die Energieversorgung. Kommunikationstechnologien und -protokolle müssen in der Gesamtsimulation als Modelle bereitgestellt werden können, um einen Kommunikationsaustausch von neuartigen IoT-Komponenten zu ermöglichen. Die Energieversorgung muss ebenfalls simuliert werden können, da hier Wechselwirkungen zwischen einzelnen Komponenten entstehen können.

Middleware, Betriebssysteme, Semantik, Security und Hardware beschränken sich hauptsächlich auf die einzelnen Komponenten selbst bzw. können ausschließlich in den Modellen der Komponenten selbst modelliert werden, welche nicht im Fokus dieser Arbeit stehen.

2.2.2 Elementare Anwendungsfälle im IoT

Neben der Objektidentifikation, beispielsweise für ein Flottenmanagement [71], lassen sich noch das Objekttracking und Sensornetzwerken als elementare Anwendungsfälle des IoT identifizieren.

Eine der wichtigsten zusätzlichen Informationen, die einem identifizierten Objekt zugeordnet werden kann, ist die Positionsinformation des Objekts. Da sich die Position eines Objekts ständig ändern kann, kann die Positionsinformation über Objekttracking gewonnen werden. Hierbei kann zwischen Outdoor-Tracking und Indoor-Tracking unterschieden werden.

Outdoor-Tracking wird schon seit Jahren durch GPS realisiert. Allerdings ist GPS oftmals nicht in Gebäuden verfügbar, da diese die Signale abschirmen, weshalb alternative Ortungstechnologien verwendet werden müssen. Eine dieser Alternativen ist die Ortung durch RFID. Hierbei werden die Objekte mit RFID-Tags ausgerüstet und sobald sie sich in der Nähe von stationären RFID-Lesegeräten befinden, kann durch die Positionsinformation des Lesegeräts auch eine Ortung des Objektes erzielt werden [72]. Auch durch andere Technologien wie zum Beispiel WLAN [73] oder Bluetooth [74] kann eine Indoor-Ortung durchgeführt werden, allerdings haben diese Technologien den Nachteil, dass die Objekte mit einer Energiequelle ausgerüstet sein müssen. Die Indoor-Ortung von Objekten erleichtert Arbeitsabläufe und reduziert Fehler durch menschliches Versagen in vielen Bereichen wie beispielsweise im Lagerwesen [75].

Durch die zunehmende Vernetzung, werden immer mehr Sensoren an das Internet angebunden, beispielsweise um Anwendungsfälle wie eine kooperative Wahrnehmung von mehreren vernetzten Fahrzeugen zu ermöglichen [76] oder autonome Roboter [77]. Dies lässt sich unter anderem auf immer billiger werdende Hardware, welche gleichzeitig immer leistungsfähiger und kleiner wird, zurückführen. Somit kann billig eine Vielzahl an Daten (Big Data) gewonnen werden, aus denen sich Informationen ableiten lassen, um Verbesserungen in Bereichen wie Betrieb, Instandhaltung, etc. zu erzielen. Gerade in diesem Bereich gab es in den letzten Jahren viele Entwicklungen. Auch im Bereich der Simulation von Sensornetzwerken wurden in letzter Zeit einige Fortschritte gemacht [78]. Durch die Anbindung von Sensoren an das Internet können unter anderem viele Remote-Anwendungen realisiert werden. So können Bauwerke durch sogenannte „Embedded Sensors“ remote-überwacht [79] oder die Instandhaltung durch Remote Monitoring [80] unterstützt werden. Bei Sensoren kann zwischen vernetzten Sensoren (Sensoren mit Kommunikationsfähigkeiten) und Smarten Sensoren, welche die Messdaten selbst interpretieren können, unterschieden werden.

Auch diese durch IoT-Technologien und -Konzepte neu entstehenden Anwendungsfälle müssen bei einem Entwurf einer IoT-Simulation berücksichtigt werden. Dies wird später in Kapitel 6.1.1 bei der Beschreibung des Evaluierungsszenarios nochmals aufgegriffen und vertieft.

2.2.3 Anwendungsbeispiel des IoT

Im Folgenden werden die Vorteile von IoT-Technologien exemplarisch im Materialmanagement eines Warenlagers aufgezeigt.

Die 6 R der Logistik besagen, dass das richtige Produkt zur richtigen Zeit am richtigen Ort in der richtigen Menge in der richtigen Qualität und zu den richtigen Kosten sein muss. Dazu gehört, dass der Bestand eines Produkts immer gewährleistet werden muss. Internet-der-Dinge-Konzepte können zur Automatisierung der Bestandskontrolle eingesetzt werden.

Schon heutzutage wird „Proactive Replenishment“ eingesetzt. Hierbei überwachen Lagerstätten wie Regale selbst welche und wie viele Waren in ihnen gelagert werden. Diese Überwachung kann beispielsweise über vernetzte Sensoren wie Drucksensoren oder Kameras durch Bildanalyse geschehen [81]. Geht eine Ware zur Neige, kann automatisch das Personal informiert [82] oder sogar autonom Nachschub bestellt werden. Zudem kann durch eine Erkennung (Identifikation) der Waren verhindert werden, dass diese falsch eingelagert werden [81]. Diese Automatisierung erhöht zudem die Verfügbarkeit der Waren, spart Zeit und vermindert die menschlichen Fehler [81]. Vor allem in der Produktion besteht durch automatisierte Bestandsüberwachung von C-Teilen, wie Schrauben oder Ähnlichem, das Potential Zeit einzusparen, da der Bestand dieser Teile nicht mehr regelmäßig durch Personal kontrolliert werden muss, sondern automatisch aufgefüllt wird. „Proactive Replenishment“ kann auch bei Verkaufsautomaten zum Einsatz kommen, wodurch die Bestände in den einzelnen Automaten nicht ständig durch Personal

kontrolliert, sondern nur bei Nachfüllbedarf angefahren werden müssen. Dieses Prinzip kann auch auf den Home-Bereich angewandt werden, indem Verbrauchsgegenstände wie Hygieneartikel automatisch nachbestellt und geliefert werden.

Der alleinige Einsatz von IoT-Technologien bietet zwar den Vorteil, dass in geschlossenen Systemen, wie beispielsweise in Warenhäusern, das Materialmanagement zwar mit herkömmlicher Automatisierungstechnik automatisiert werden kann, durch Medienbrüche geht diese Automatisierung allerdings nur bis zu den Grenzen dieser Systeme. Durch Internet-der-Dinge-Konzepte hingegen kann die Automatisierung über diese Grenzen hinaus fortgeführt werden. Bei Verkaufsautomaten, welche örtlich verteilt aufgestellt werden, ist eine Vernetzung dieser unumgänglich, falls „Proactive Replenishment“ angewendet werden soll.

Die Vernetzung von Gegenständen bringt allerdings nicht nur Vorteile, wenn sie in einem bestimmten Anwendungsbereich eingesetzt wird, sondern weitere Vorteile können durch eine Vernetzung über Domänengrenzen hinaus erzielt werden, da sich die Interoperabilität zwischen den Systemen erhöht.

Dieses kurze Szenario zeigt, dass IoT-Systeme einige Vorteile gegenüber herkömmlichen Systemen bringen, allerdings wird auch ersichtlich, dass durch die hohe Vernetzung und Dezentralisierung die Komplexität solcher Systeme steigt und es immer schwieriger wird, ein derartiges System komplett ohne Modelle und Simulationen zu analysieren. Dieses Szenario wird in Kapitel 6.1 nochmals aufgegriffen, um ein Evaluierungsszenario für das erstellte Konzept zu generieren. Im nächsten Unterkapitel werden grundlegende Simulationskonzepte beschrieben, mit denen solche Szenarien potenziell simulierbar sind.

2.3 Bestehende Ansätze zur Simulation von IoT-Systemen

In der Literatur existieren bereits mehrere Ansätze zur Simulation von IoT-Systemen. Ein Resultat einer Literaturrecherche [83] im Rahmen dieser Arbeit ist, dass die Aspekte der Simulation, auf die sich die einzelnen Veröffentlichungen konzentrieren, stark variieren, siehe Tabelle 1 und Tabelle 2. Die in diesem Kapitel beschriebenen Ansätze, sind alles Ansätze, welche nur ein Simulationstool enthalten, mit dem das gesamte System simuliert wird. Hierbei unterteilen sich die Ansätze in Diskrete-Event-Simulationen und agentenbasierte Simulationen, weiter Simulationsarten mit denen IoT-Systeme simuliert werden, konnten bei der Literaturrecherche nicht gefunden werden.

Ein Aspekt, auf den sich die Simulationskonzepte fokussieren, ist z.B. eine detaillierte Simulation der Kommunikation zwischen den IoT-Komponenten. Hierbei werden Kommunikationsprotokolle implementiert und physikalische Aspekte wie Interferenzen berücksichtigt. Ein anderer Aspekt ist die Skalierbarkeit der Simulation, speziell in Wireless Sensor Networks (WSN). So ist es mit diesen Ansätzen möglich bis zu mehrere Millionen

Komponenten zu simulieren, welche miteinander interagieren und kommunizieren. Viele der untersuchten Ansätze konzentrieren sich auf spezifische Domänen und Anwendungsfälle wie beispielsweise Gebäudeautomatisierung oder Energieverbrauch der IoT-Komponenten und sind nicht auf andere Domänen übertragbar.

Im Folgenden werden die einzelnen gefundenen Ansätze, unterteilt in Diskrete-Event-Simulatoren und agentenbasierte Simulatoren, detailliert vorgestellt.

2.3.1 Diskrete-Event-Simulatoren

Zuerst wird eine kurze Beschreibung von Diskreter-Event-Simulation gegeben und anschließend werden die untersuchten Diskrete-Event-Simulatoren vorgestellt und abschließend diskutiert.

2.3.1.1 Diskrete-Event-Simulation

Diskrete-Event-Simulation (DES) wird zur Simulation von Diskrete-Event-Systemen eingesetzt. Diskrete-Event-Systeme sind dynamische Systeme, die ihren Zustand nur zu bestimmten Zeitpunkten (Events) ändern. Zwischen diesen Zeitpunkten ändert sich der Zustand des Systems, im Gegensatz zu kontinuierlichen Systemen, nicht [84]. Somit gibt es auch nur zu diesen bestimmten Zeitpunkten einen Simulationsfortschritt, da zwischen diesen Zeitpunkten eine Simulation nicht benötigt wird, da sich das System zu diesen Zeitpunkten ebenfalls nicht ändert [85]. Tritt ein Ereignis auf, so wird der dazugehörige Code (Ereignisroutine) in der Simulation ausgeführt [86]. Ereignisse können hierbei voneinander abhängig sein oder völlig unabhängig auftreten [87]. Die Verwaltung der Ereignisse und der Simulationszeit erfolgt durch einen Steuerungsalgorithmus, welcher zentral und unabhängig von der Anwendung ausgeführt wird [86]. Es gibt zwei Möglichkeiten die Simulationszeit fortschreiten zu lassen. Einerseits kann die Simulation in konstanten Zeitintervallen fortschreiten. Eine Schwierigkeit hierbei ist die geeignete Wahl dieses Zeitintervalls. Bei einem zu großen Zeitintervall nimmt die Genauigkeit der Simulation ab, ein zu kleines Zeitintervall erhöht die Berechnungszeit der Simulation unnötig [39], [88]. Andererseits können die Zeitschritte auch unregelmäßig sein, wodurch immer zum nächsten Event gesprungen wird. Hier stellt sich aber die Frage, wie diese Zeitschritte bestimmt werden. Hierbei müssen auch Events berücksichtigt werden, die eine gewisse Zeit brauchen, um das nächste Zeitintervall richtig berechnen zu können [89].

Diskrete-Event-Simulation wird für Kommunikationsnetze eingesetzt [90], da in einem Kommunikationsnetzwerk Nachrichten zu bestimmten Zeitpunkten versendet werden. Somit kann sie auch für IoT-Systeme eingesetzt werden, wenn die Kommunikation der einzelnen Komponenten bei dieser Simulation im Vordergrund steht.

2.3.1.2 IoT-Simulation mit Diskreter-Event-Simulation

Einige existierende Ansätze setzten die Interaktion zwischen den einzelnen Komponenten mithilfe von DES um. Hierbei wurden meistens existierende Simulationstools erweitert und angepasst, um den Anforderungen zu genügen. Diese Ansätze werden im Folgenden jeweils kurz beschrieben.

[91] setzt eine DES um und fokussiert sich auf die Skalierbarkeit von IoT-Systemen. Dieser Ansatz betrachtet nur IoT-Szenarien in urbanen Umgebungen, inklusive des Energieverbrauchs der IoT-Geräte. In urbanen Umgebungen bewegen sich die meisten Komponenten wie beispielsweise Kraftfahrzeuge. Um die Simulation solcher Komponenten zu realisieren, teilt sich das Modell einer Komponente in ein Mobilitätsmodell, welches die Bewegungen der Komponente umsetzt, ein Netzwerkmodell, welches die Kommunikation zwischen den Komponenten realisiert, und ein Energiemodell, welches den Energieverbrauch der Komponente beschreibt. Cooja, ein Netzwerksimulator, wurde für die Energie- und Netzwerkmodelle verwendet. Die Netzwerkmodelle wurden noch durch ns-3, einem DES-Netzwerk-Simulator für Internetsysteme, erweitert. OSMobility, ein Verkehrssimulator, wurde verwendet, um die Mobilitätsmodelle umzusetzen. Die Interaktionen zwischen den einzelnen Komponenten wurden mit DEUS (Discrete Event Universal Simulator) realisiert. ns-3 ermöglicht eine detaillierte Simulation der Kommunikationskanäle. Mit dem vorgestellten Simulator können bis zu 50.000 Komponenten simuliert werden.

DPWSim simuliert IoT-Systeme, um die Entwicklung von Anwendungen, die Devices-Profile-for-Web-Services (DPWS) verwenden, zu erleichtern und wird in [92] vorgestellt. Der vorgestellte Simulator ist ein Diskreter-Event-Simulator und simuliert Events, welche von den Funktionen der modellierten Geräte getriggert werden. Das Modell eines Geräts besteht aus „Spaces“, welche die Umgebung des Geräts repräsentieren, „Operations“, welche die Funktionalitäten eines Geräts beschreiben und „Devices“. Das Ziel dieses Simulators ist es nicht, die Entwicklung von IoT-Systemen zu unterstützen, sondern die Entwicklung von Anwendungen die im IoT laufen.

IoTSim [93] ist ein Simulator, welcher zur Generierung von Big Data verwendet wird, welche Big Data ähneln, die von IoT-Systemen erzeugt werden. Diese Daten können verwendet werden, um Anwendungen, die Cloud-Computing verwenden, zu entwickeln und zu testen. IoTSim ist ein Diskreter-Event-Simulator, die Modelle bestehen aus Entitäten und Events. Entitäten repräsentieren Komponenten, welche unabhängig von anderen Komponenten existieren und Nachrichten versenden können. Die Entitäten können Events triggern und auf die Events anderer Entitäten reagieren. Ein Event findet zwischen mindestens zwei Entitäten statt.

In [94] wird das Simulationstool MAMMOTH vorgestellt, welches in der Lage ist mehrere Millionen von Wireless Sensoren zu simulieren. Der Simulator verwendet Diskrete-Event-Simulation, um TinyOS (Betriebssystem für Sensornetzwerke) Geräte zu emulieren. Er kann

ebenfalls genutzt werden, um die Hardware zu simulieren, auf der Anwendungen ausgeführt werden, die auf Sensoren laufen.

[95] fokussiert sich auf die Simulation von Smart-Home-Szenarien und verwendet Diskrete-Event-Simulation. Das Ziel ist es hierbei, die Kommunikation zwischen den Komponenten unter Berücksichtigung von physikalischen Effekten zu simulieren. Durch eine Simulation der Umgebung der Geräte können Interferenzen und Wellenausbreitungen simuliert werden. Zusätzlich wird Monte-Carlo-Simulation verwendet, um eine statistisch verteilte Kommunikation zwischen den Geräten zu erzeugen.

[96] beschreibt Methoden zur Modellierung von Cyber-Physischen Systemen. Für die Modellierung wird Modelica, eine Modellierungssprache, verwendet. Die Modelle der Cyber-Physischen Systeme sind in diskrete und kontinuierliche Modelle aufgeteilt. Im diskreten Modell werden Aspekte wie die Kommunikationsfähigkeiten des Cyber-Physischen Systems modelliert, während im kontinuierlichen Modell physikalische Aspekte beschrieben werden, die kontinuierlich ablaufen, wie beispielsweise die Bewegung eines Roboterarms. Für die Synchronisation zwischen den Modellen (diskretes und kontinuierliches) agiert das diskrete Modell als Master und synchronisiert sich mit dem kontinuierlichen Modell zu diskreten Zeitpunkten.

[97] simuliert ebenfalls Wireless Sensor Networks. Für die Modellierung wird eine Kombination aus OMNeT++ und Cooja verwendet. OMNeT++ ist ein Simulationstool zur Diskrete-Event-Simulation von Kommunikationsnetzwerken, in welchem jeder Kommunikationsteilnehmer durch einen Knoten dargestellt wird. OMNeT++ wird hier zur Modellierung der Kommunikationsaspekte der Sensoren verwendet und Cooja simuliert die Funktionen und das Betriebssystem der Sensoren.

In [98] werden mit Hilfe von Diskreter-Event-Simulation Szenarien im Bereich der Gesundheitspflege simuliert. In diesem Ansatz wird Echtzeit berücksichtigt und die verwendeten Modelle sind in drei Schichten unterteilt: eine Entitätsschicht, eine Verhaltensschicht und eine Schicht zur Performancesimulation. Dieser Simulator wird zur Simulation von Anwendungen eingesetzt, die Cloud-Computing verwenden.

[99] verwendet Cooja um virtuelle IoT-Devices für Gebäudemodelle sowie eine Umgebung für diese zu simulieren. Neben den Gebäudemodellen geht es hierbei hauptsächlich darum simulierte Sensorwerte von IoT-Devices zur Verfügung zu stellen.

In [100] werden IoT-Netzwerke mit mobilen IoT-Komponenten simuliert. Es wird angenommen, dass die mobilen IoT-Komponenten mit stationären Knotenpunkten kommunizieren, die diese Nachrichten weiterleiten. Ziel der Simulation ist es, die Auslastung dieser Knotenpunkte zu ermitteln.

[101] simuliert IoT-Netzwerke mit begrenzter Bandbreite. Hierzu nutzt es den Simulator OPNET und konzentriert sich auf Netzwerke, die Long-Term Evolution nutzen und über eine große Fläche verteilt sind.

[102] simuliert IoT-Netzwerke mit begrenzter Bandbreite und IoT-Systeme, welche Long-Term Evolution nutzen. Es wird ebenfalls OPNET benutzt.

In Tabelle 1 sind die einzelnen Ansätze, die Diskrete-Event-Simulation verwenden, zur Simulation von IoT-Szenarien sowie der Hauptaspekt auf den sich die Simulation konzentriert zusammengefasst.

Tabelle 1: Übersicht der Diskrete-Event-Simulationsansätze

Ansatz	**Verwendete Modellierungstools**	**Fokus der Simulation**
[91]	DEUS, Cooja, ns-3	Skalierbarkeit von IoT-Systemen
[92]	DPWSim	Entwicklung von IoT-Anwendungen
[93]	Cloudsim, IoTSim	Big-Data-Verarbeitung aus Sensornetzwerken
[94]	MAMMOTH	Betrieb von Wireless Sensor Networks
[95]	-	Simulation der Kommunikation in IoT-Systemen
[96]	Modelica	IoT-Systeme mit Cyber-Physischen Komponenten
[97]	Cooja, OMNeT++	Betrieb von Wireless Sensor Networks
[98]	SimIoT, SimIC	Cloud-Computing von IoT-Systemen
[99]	Cooja, Contiki	Simulation von einzelnen Komponenten
[100]	-	Netzwerkperformance von IoT-Systemen
[101]	OPNET	Narrow-Band-Übertragung in IoT-Systemen
[102]	OPNET	Narrow-Band-Übertragung in IoT-Systemen

2.3.1.3 Diskussion der Diskrete-Event-Simulationsansätze

Wie Tabelle 1 zeigt, fokussieren sich alle diese Ansätze auf einen Aspekt und blenden viele andere relevante Aspekte dadurch aus. Bei diesen Ansätzen bietet sich meistens nicht die Möglichkeit, Modelle zur Laufzeit in die Simulation zu integrieren. Im Speziellen ist es bei allen nicht möglich, Modelle, die zum Start der Simulation noch nicht bekannt sind, zu integrieren. Auch bieten diese Ansätze nicht die Möglichkeit, mehrere Simulationstools zur Simulation der einzelnen Komponenten zu verwenden. [83]

Grundsätzlich kann für die Simulation der Kommunikation zwischen IoT-Komponenten Diskrete-Event-Simulation verwendet werden. Allerdings ist es schwierig ein dynamisches, heterogenes

IoT-System mit einer reinen Diskrete-Event-Simulation zu simulieren. Eine Diskrete-Event-Simulation wird zentral koordiniert [86]. IoT-Systeme hingegen können auch dezentral organisiert sein, wodurch es schwierig wird die Dezentralität in der Simulation abzubilden. Zudem ändert sich bei Eintritt einer neuen Komponente die Reihenfolge der Events, wodurch bei jedem Eintritt die Planung der Events neu durchgeführt werden muss. Es ist auch nicht möglich, alle Komponenten mit einer Diskrete-Event-Simulation zu simulieren, da in den Komponenten kontinuierliche Prozesse ablaufen können, für deren Simulation eine Diskrete-Event-Simulation ungeeignet ist. Diese Ergebnisse haben auch Untersuchungen von einem typischen Diskrete-Event-Simulationstool (SimEvents) im Rahmen dieser Arbeit gezeigt.

Untersuchungen von typischen Diskrete-Event-Simulationstools (VEINS, VSimRTI, OMNet++ und MATLAB-Simulink sowie MATLAB-Statflow) haben ebenfalls gezeigt, dass es mit diesen Simulationstools nicht, beziehungsweise nur sehr schwer möglich ist, den Eintritt von Komponenten in ein IoT-System zu simulieren. Hierzu wurden typische IoT-Szenarien simuliert, im speziellen, eine Verkehrskreuzung an der die unterschiedlichen Verkehrsteilnehmer untereinander sowie mit der Verkehrsinfrastruktur kommunizierten. Zudem traten neue IoT-Komponenten in Form von Verkehrsteilnehmern ein und aus. Zudem wurde ein intelligentes Warenlager simuliert, in das kommunikationsfähige Waren ein- und austraten. Allerdings waren Eintritte von Komponenten die zum Startzeitpunkt der Simulation nicht bekannt waren, nicht simulierbar, da alle Modelle vor dem Start der Simulation integriert sein müssen. Somit werden weder die Anforderung der Dynamik (A4) noch die Anforderung der Heterogenität (A3) erfüllt. Bei der Erstellung der Simulationen unterstützen studentische Arbeiten, die Schlüsse, was dies für die Simulation von IoT-Systemen unter Berücksichtigung der genannten Anforderungen bedeutet wurden aber eigenständig im Rahmen dieser Arbeit gezogen. Weiterführende Informationen befinden sich im Anhang.

2.3.2 Agentenbasierte Simulatoren

Zuerst wird eine kurze Beschreibung von agentenbasierter Simulation gegeben und anschließend werden die untersuchten agentenbasierten Simulatoren vorgestellt und abschließend diskutiert.

2.3.2.1 Agentenbasierte Simulation

Eine Simulation mit agentenbasierter Modellierung ist im Gegensatz zur Diskrete-Event-Simulation dezentral aufgebaut [103]. In der agentenbasierten Simulation werden die einzelnen Bestandteile eines Systems mit ihrem entsprechenden Verhalten modelliert und das Verhalten des Gesamtsystems ergibt sich aus der Interaktion der einzelnen Bestandteile. Somit verfolgt die agentenbasierte Simulation im Gegensatz zur Diskrete-Event-Simulation einen Bottom-Up-Ansatz [103]. Bei der agentenbasierten Simulation werden aktive Objekte des realen Systems als aktive Agenten modelliert [104]. Diese Agenten können genauso wie die realen Objekte mit ihrer Umwelt interagieren und diese mit Sensoren erfassen [105]. Hierbei verfolgt jeder Agent eigene

Aktionen, welche von seiner Umwelt, seinem Zustand und den Interaktionen mit anderen Agenten abhängen [106]. Durch die individuelle Modellierung der einzelnen Objekte können variable Interaktionen zwischen den einzelnen Objekten unterstützt werden [107]. Agentenbasierte Simulation findet in vielen Gebieten Anwendung, allerdings eignet sie sich besonders gut für komplexe Systeme, welche nicht oder nur sehr schwer zentral beschrieben werden können [108], [109]. Durch die dezentrale Beschreibung der Systeme können emergente Phänomene simuliert werden, welche durch das Verhalten einzelner Objekte ausgelöst werden und eine Auswirkung auf das gesamte System haben [110]. Zudem können durch die individuelle Modellierung der einzelnen Objekte sehr unterschiedliche und heterogene Systeme modelliert werden [105]. Durch diese individuelle Modellierung ergibt sich allerdings ein sehr hoher Aufwand bei der Erstellung der Modelle, so dass abgeschätzt werden muss, ob der erhöhte Aufwand gerechtfertigt ist [104]. Da IoT-Systeme oftmals dezentrale Systeme sind, bietet sich die agentenbasierte Simulation für IoT-Systeme an. Hierbei können die einzelnen IoT-Komponenten durch Agenten modelliert werden und das Verhalten des IoT-Systems resultiert aus dem Verhalten der einzelnen Komponenten. Somit ist die agentenbasierte Simulation bei der Simulation von dezentralen Systemen wesentlich realitätsnaher als die Diskrete-Event-Simulation [111]. Die Dynamik von IoT-Systemen, in denen sich die Beziehungen zwischen den Komponenten durch das Ein- und Austreten von Komponenten ständig ändern, lässt sich durch agentenbasierte Simulation abbilden, da auch die Beziehungen zwischen Agenten dynamisch modellierbar sind [112].

2.3.2.2 IoT-Simulation mit agentenbasierter Simulation

Auch bei agentenbasierten Simulatoren wurden meistens existierende Simulationstools erweitert und angepasst, um den Anforderungen zu genügen. Diese Ansätze werden im Folgenden jeweils kurz beschrieben.

[113] verwendet agentenbasierte Simulation und fokussiert sich auf eine detaillierte Simulation der Kommunikation zwischen den IoT-Komponenten, um hierbei Engpässe zu entdecken. Diese detaillierte Simulation beinhaltet Round-Trip-Times und Packet-Delivery-Rates. Die modellierten Komponenten sind smarte Objekte, welche sehr ähnlich zu Cyber-Physischen Komponenten sind. Jedes smarte Objekt wird durch einen Agenten repräsentiert und die Kommunikation zwischen diesen wird durch OMNeT++ realisiert. Im beschriebenen Ansatz wird jeder Knoten als Agent umgesetzt, wodurch die Autonomie der smarten Objekte gewährleistet wird. In diesem Fall wurde ein Diskretes-Event-Simulationstool zu einem Simulationstool zur agentenbasierten Simulation erweitert.

In [114] wird ebenfalls ein agentenbasiertes Simulationskonzept beschrieben, welches sich auf die Kommunikation zwischen den Komponenten mithilfe von DPWS konzentriert. DPWS ermöglicht es Geräte mit eingeschränkten Ressourcen Web Services zu verwenden. DPWS hat allerdings die Einschränkung, dass alle Geräte SOA-ready (Service Oriented Architecture) sein müssen. In diesem Konzept ist es möglich, reale Geräte in die Simulation zu integrieren, da diese

nicht von simulierten Geräten zu unterscheiden sind. Die Kommunikation zwischen den Agenten, welche die DPWS-Geräte repräsentieren, wird durch ACL (Agent Communication Language) bzw. DPWS und die Kommunikation zu den realen Geräten durch DPWS realisiert.

In [115] werden Wireless Sensor Networks simuliert, wodurch auch die Skalierung von IoT-Systemen mitberücksichtigt wird. Die örtliche Verteilung der Sensoren wird durch OpenStreetMap, einer frei nutzbaren Geodatensammlung, realisiert. Der Simulator ist agentenbasiert und besteht aus mehreren Modulen:

- Agentenmodul: Dieses Modul besteht aus Devices und Events, welche von den Devices getriggert werden können.
- OpenStreetMap-Modul: Dieses Modul wird für die örtliche Verteilung der Sensoren verwendet.
- WiSen-Simulator-Modul: Dieses Modul wird zu Koordination der Simulation verwendet.
- Solver-Modul: Dieses Modul optimiert die für die Simulation benötigten Berechnungen.

Die Agenten, die die Geräte repräsentieren, können durch Skriptdateien konfiguriert werden und triggern Events, welche ebenfalls in den Skriptdateien beschrieben sind. In der nächsten Version des Simulators soll eine Interaktion mit der Umgebung der Sensoren enthalten sein. Dadurch wird es möglich meteorologische Ereignisse wie Temperatur und Winde zu simulieren.

SenseSim [116] ist ein agentenbasierter Simulator für Wireless Sensor Networks. In diesem Simulator wird die Kommunikation idealisiert und die unteren Schichten des OSI-Modells werden nicht implementiert. Der Hauptfokus liegt auf der Interaktion der Sensoren mit ihrer Umwelt. Die Sensoren werden durch Agenten repräsentiert, welche aus vier Klassen bestehen:

- SensorLogic: Methoden, die die Zustände der Sensoren ändern
- SimEntity: Die Hauptklasse, die den Sensor verwaltet
- SensorAPI: Basisfunktionen, die der Sensor benötigt, wie das Versenden von Nachrichten
- Middleware: Management zur Ausführung des Programms, das auf dem Sensor läuft

Es ist zudem möglich reale Sensoren in die Simulation zu integrieren, da sich die simulierten Sensoren gleich verhalten und somit reale Sensoren ersetzen können.

Der Fokus von [117] liegt auf der agentenbasierten Simulation von Smart-Grid-Systemen. In diesem Ansatz werden nur der Energieverbrauch und die Energieerzeugung von Geräten simuliert. Die Simulation wird in Fast-Echtzeit durchgeführt und jedes Gerät wird durch einen Agenten repräsentiert. Es ist eine Interaktion mit realen Geräten möglich und zudem kann der Benutzer zur Simulationszeit mit den existierenden Agenten interagieren. Er kann auch zur

Simulationszeit Agenten hinzufügen bzw. entfernen. Allerdings ist es nur möglich solche Agenten zu erzeugen, die ein bereits im Simulator bestehendes Modell repräsentieren.

[118] konzentriert sich auf eine parallele und verteilte Simulation, um eine hohe Skalierbarkeit des simulierten Systems zu erreichen und trifft die Annahme, dass in einem großen System nicht alle Teile des Systems mit einem hohen Detaillierungsgrad simuliert werden müssen, sondern dass es ausreicht, die interessanten Bereiche detailliert zu simulieren und bei den restlichen Bereichen eine gröbere Simulation zu verwenden. Hierfür wird Multilevel-Modelling verwendet, bei dem verschiedene Modelle für die verschiedenen Bereiche mit einem unterschiedlichen Detaillierungsgrad verwendet werden. Alle Modelle verwenden Diskrete-Event-Simulation. Die detaillierteren Modelle verwenden mehr Zeitschritte als die gröberen Modelle und müssen mit den gröberen Modellen synchronisiert werden. Diese Synchronisation geschieht zu den Zeitschritten, in denen das gröbste Modell seinen Zustand ändert.

[119] beschreibt einen agentenbasierten Simulator, der DPWS verwendet und damit der Einschränkung unterliegt, dass alle Geräte SOA-ready sein müssen. Jeder Agent repräsentiert eine Komponente und es ist möglich komplexe Komponenten aus einfachen Komponenten aufzubauen. Eine Schicht über den Agenten befindet sich eine Logikschicht, die die Szenarien verwaltet, die der Nutzer mit der Simulation ausführen möchte.

In [120] werden ebenfalls Sensornetzwerke simuliert, hierbei geht es hauptsächlich im die Skalierbarkeit von solchen Sensornetzwerken. Zum Einsatz kam das Simulationstool Ptolemy II und ein MQTT-Broker zur Kommunikation zwischen den einzelnen simulierten Knoten.

[121] versucht sowohl das Verhalten von IoT-Devices als auch von Nutzern dieser Devices zu simulieren. Hierbei steht der Test von einzelnen IoT-Devices im Vordergrund und es werden reale Daten zu diesen Simulationen herangezogen.

In [122] werden IoT-Devices mit ACOSO agentenorientiert modelliert und auf OMNet aufgesetzt, welches die Kommunikation simuliert und Diskret-Event-basiert ist. Fokus ist hier die Performance der IoT-Devices.

In Tabelle 2 sind die einzelnen Ansätze, die agentenbasierte Simulation zur Simulation von IoT-Szenarien verwenden, sowie der Hauptaspekt auf den sich die Simulation konzentriert zusammengefasst.

Tabelle 2: Übersicht der agentenbasierten Simulationsansätze

Ansatz	Verwendete Modellierungstools	Fokus
[113]	OMNeT++	Simulation der Kommunikation in IoT-Systemen
[114]	OMNeT++	Simulation der Kommunikation über Devices Profile for Web Services
[115]	CupCarbon	Betrieb von Wireless Sensor Networks
[116]	SenseSim	Betrieb von Wireless Sensor Networks
[117]	-	Energieverbrauch von IoT-Komponenten
[118]	GAIA/ARTÌS, OMNeT++, ns-3	Skalierbarkeit von IoT-Systemen
[119]	-	Simulation der Kommunikation über Devices Profile for Web Services
[120]	Ptolemy II	Skalierbarkeit von IoT-Systemen
[121]	-	Test von IoT-Komponenten
[122]	ACOSO, OMNet	Performance von IoT-Systemen

2.3.2.3 Diskussion der agentenbasierten Simulationsansätze

Die im vorherigen Kapitel beschriebenen Veröffentlichungen wurden auf ihre Tauglichkeit zur Simulation von IoT-Systemen hin untersucht. Die Kriterien die für diese Untersuchung angewandt wurden, sind die in Kapitel 1.3 hergeleiteten Anforderungen, im Einzelnen:

- Können Modelle hinzugefügt oder entfernt werden, ohne dass das Gesamtmodell manuell angepasst werden muss?
- Können Modelle hinzugefügt oder entfernt werden, ohne dass die möglichen Interaktionen mit anderen Modellen und deren Simulationen manuell angepasst werden müssen?
- Können Teilmodelle in unterschiedlichen Simulationstools aus unterschiedlichen Domänen simuliert werden?
- Kann die Gesamtsimulation zur Laufzeit um weitere Teilsimulationen erweitert werden?

Jede der oben aufgeführten Veröffentlichungen wurde auf diese Kriterien hin untersucht und es wurde bewertet, in welchem Umfang jede dieser Veröffentlichungen welches dieser Kriterien erfüllt. Hierbei hat sich gezeigt, dass bei einigen dieser Ansätze sich die Möglichkeit neue Modelle zur Laufzeit der Simulation zu erzeugen bietet. Allerdings ist es bei keinem Ansatz möglich Modelle, die zum Start der Simulation noch nicht bekannt sind, zu integrieren. Auch bieten diese

Ansätze nicht die Möglichkeit, mehrere Simulationstools zur Simulation der einzelnen Komponenten zu verwenden. [83]

Darüber hinaus hat sich gezeigt, dass die agentenbasierte Simulation durch ihre dezentrale Organisation einem realen IoT-System wesentlich näherkommt als eine Diskrete-Event-Simulation. Auch das Eintreten von neuen Komponenten kann mit einer agentenbasierten Simulation wesentlich einfacher realisiert werden als bei der Diskrete-Event-Simulation. Zudem bietet die klassische agentenbasierte Simulation nicht die Möglichkeit, verschiedene, spezialisierte Simulationstools zu verwenden, da in der klassischen agentenbasierte Simulation nur eine Modellierungsmethode zur Modellierung der Komponenten angewandt wird.

Diese Ergebnisse haben auch Untersuchungen anhand von mehreren typischen agentenbasierten Simulationstools (SeSAm, Swarm, AgentSheets, MadKit, NetLogo und Simio) gezeigt. Bei diesen Untersuchungen wurden zwei typische IoT-Szenarien (IoT-basierte Ampelsteuerung an einer Kreuzung und eine IoT-basiertes Warenlagers) simuliert. Hierbei wurde der grundlegende Ablauf dieser Szenarien simuliert, ein besonderer Schwerpunkt lag hierbei auf der Simulation des Ein- und Austretens von Komponenten (Fahrzeuge im Fall der Ampelsteuerung und Waren sowie Gabelstapler im Fall des Warenlagers). Diese Untersuchungen haben gezeigt, dass es mit diesen Simulationstools zwar möglich ist den Eintritt von Komponenten in ein IoT-System zu simulieren, allerdings ist es mit ihnen nicht möglich unterschiedliche Modelle zu verwenden. Somit erfüllen herkömmliche agentenbasierte Simulationstools nicht die Anforderungen der Heterogenität an die Simulation von IoT-Systemen. Auch konnten keine neuen, zur Entwurfszeit unbekannten Modelle zur Laufzeit in die Simulation eingebunden werden wodurch „Plug-and-Simulate“ mit diesen Simulationstools ebenfalls nicht möglich ist. Somit wird die Anforderung der Dynamik (A4) zwar ansatzweise erfüllt, allerdings wird die Anforderung der Heterogenität (A3) ebenfalls nicht erfüllt. Bei der Erstellung der Simulationen unterstützen studentische Arbeiten, die Schlüsse, was dies für die Simulation von IoT-Systemen unter Berücksichtigung der genannten Anforderungen bedeutet wurden aber eigenständig im Rahmen dieser Arbeit gezogen. Weiterführende Informationen befinden sich im Anhang.

2.3.3 Zusammenfassung

Die Analyse der Diskrete-Event-Simulationstools sowie der agenten-basierten Simulationsansätze zur Simulation von IoT-Szenarien hat gezeigt, dass keines dieser vorhandenen Simulationskonzepte, die in Kapitel 1.3 hergeleiteten Anforderungen komplett erfüllt. Diese Analysen wurden durch Untersuchungen von typischen Simulatoren anhand von repräsentativen IoT-Szenarien bestätigt. Keines der untersuchten Simulationskonzepte sieht eine einfache Integration neuer, zur Entwurfszeit unbekannter Modelle zur Laufzeit der Simulation vor (A4), auch wenn zur Entwurfszeit bekannte Modelle zur Laufzeit in agentenbasierte Simulatoren prinzipiell eingefügt werden können. Die Verwendung von verschiedenen Simulationstools, zur optimierten Modellierung der Komponenten ist mit diesen Ansätzen ebenfalls nicht möglich (A3).

Die Diskrete-Event-Simulatoren erfüllen zudem weder die Anforderungen der modularen Modellierung (A1) noch die Anforderung der Integration ohne Anpassung der Interaktionen (A2). Die agentenbasierten Simulatoren hingegen erfüllen diese beiden Anforderungen, solange die Modelle zur Entwurfszeit der Simulation bekannt sind. Diese Ergebnisse wurden ebenfalls nochmal in mehreren intensive Diskussionen mit mehreren Experten eines Industriepartners im Rahmen eines Industrieprojekts bestätigt.

Tabelle 3 fasst diese Erkenntnis zusammen.

Tabelle 3: Bewertung der Diskrete-Event- und agentenbasierten Simulatoren hinsichtlich der Anforderungen an eine Simulation eines IoT-Systems

Anforderung / **Ansatz**	**A1**	**A2**	**A3**	**A4**
Diskrete-Event-Simulatoren	○	○	○	○
Agentenbasierte Simulatoren	●	●	○	◑

● Erfüllt ◑ Teilweise Erfüllt ○ Nicht Erfüllt

Dies zeigt, dass bisher noch kein Ansatz zur Simulation von IoT-Szenarien entwickelt wurde, der die in Kapitel 1.3 gestellten Anforderungen erfüllt, und dass alle Ansätze die Anforderung der Heterogenität (A3) nicht erfüllen. Es ist grundsätzlich sehr schwierig bzw. nahezu unmöglich in einem einzelnen Simulator der Anforderung der Heterogenität (A3) gerecht zu werden, da dieser Simulator sehr viel Domänen und sehr viele Aspekte von IoT-Systemen abbilden können müsste. Daher werden im nächsten Kapitel Ansätze zur dynamischen Co-Simulation von Systemen, welche mehrere Simulationstools koppeln, anderer Anwendungsbereiche untersucht. Insbesondere in Hinsicht darauf, welche die gestellten Anforderungen teilweise oder komplett erfüllen könnten.

2.4 Bestehende Ansätze zur dynamischen Co-Simulation

In diesem Unterkapitel werden bestehende Standards und Ansätze zur Co-Simulation untersucht. Co-Simulationen ermöglichen es, unterschiedliche Teilsimulationen zu einer Gesamtsimulation zu verknüpfen. Somit können einzelne Aspekte eines zu simulierenden Systems in spezialisierten Simulationstools simuliert werden, wodurch es nicht ein Simulationstool benötigt, was alle erforderlichen Aspekte simulieren kann. Dies ermöglicht eine deutlich genauere und effizientere Simulation. Diese Aspekte können unterschiedlich geartet sein, so ist es beispielsweise möglich, einzelne Subsysteme getrennt voneinander zu simulieren oder unterschiedliche physikalische Aspekte getrennt voneinander zu simulieren.

Zuerst wird eine Einführung in Co-Simulation gegeben und anschließend bestehende Co-Simulationsansätze vorgestellt. Diese entsprechen dem Stand der Technik, da sie bereits seit mehreren Jahren in der Industrie eingesetzt werden und teilweise sogar standardisiert sind. Hierbei wurden domänenspezifische Standards, das Functional Mock-up Interface sowie High Level Architecture untersucht. Anschließend werden Co-Simulationsansätze aus dem Stand der Forschung untersucht, allgemeine Ansätze aus der Literatur sowie Co-Simulationen basierend auf einer Service-orientierten Architektur, basierend auf OSGi und basierend auf einem Agentensystem. Im folgenden Unterkapitel werden diese Ansätze auf die Erfüllung der in Kapitel 1.3 gestellten Anforderungen hin untersucht.

2.4.1 Co-Simulation

Es ist nicht möglich jegliche Komponenten mit allen erdenklichen Aspekten in einem Simulationstool zu simulieren. Es werden für die Simulation eines Systems oder einer Komponente für bestimmte Aspekte spezialisierte Simulationstools benötigt [29], Zulieferer verwenden spezialisierte Simulationstools [123] oder in den Modellen der Komponenten ist geistiges Eigentum enthalten, weshalb es nicht in ein anderes Simulationstool übertragbar ist [124]. Bei solchen Problemen werden die unterschiedlichen Aspekte in unterschiedlichen Simulationstools simuliert und die Simulationstools gekoppelt, um die Abhängigkeiten zwischen den Teilaspekten zu realisieren [125]. Diese Kopplung von unterschiedlichen Simulationstools wird als Co-Simulation (cooperative simulation) bezeichnet. Beispiele für den Einsatz von Co-Simulation sind multiphysikalische Simulationen [30] und Simulationen, bei denen sowohl Hardware als auch Software simuliert wird [117]. Somit kann Co-Simulation für die Simulation einzelner Komponenten mit verschiedenen Aspekten eingesetzt werden. Co-Simulation kann aber auch für die Simulation von Systemen mit mehreren Komponenten eingesetzt werden, die individuell modelliert werden [126].

Eine Co-Simulation kann aus zwei oder mehr gekoppelten Simulationstools bestehen [127]. Besteht eine Co-Simulation nur aus zwei Simulationstools, so können diese direkt aufeinander abgestimmt werden. Oftmals werden aber mehr als zwei Simulationstools benötigt, weshalb eine direkte Abstimmung nicht immer wünschenswert ist, um eine leichte Erweiterung um weitere Simulationstools zu ermöglichen. Daher werden Verfahren für eine effiziente Kopplung von mehreren Simulationstools benötigt [128].

Co-Simulation lässt sich in iterative Kopplung und in nichtiterative Kopplung unterteilen [129]. Bei der iterativen Kopplung werden die einzelnen Simulationen mehrfach für jeden Zeitschritt ausgeführt und nach jeder Iteration miteinander abgeglichen, bis ein Abbruchkriterium erreicht wurde [130]. Die nichtiterative Kopplung kann noch in parallele und sequentielle Co-Simulation unterteilt werden [129]. Die parallele Co-Simulation wird als Jacobi-Schema bezeichnet und die sequentielle Co-Simulation als Gauß-Seidel-Schema [131]. Bei dem Jacobi Schema werden für jeden Zeitschritt die einzelnen Teilsimulationen parallel berechnet und danach werden die

Koppelgrößen ausgetauscht [131]. Bei dem Gauß-Seidel-Schema wird zuerst eine Teilsimulation für den Zeitschritt n berechnet und deren Koppelgrößen als Eingangsgrößen für die Berechnung der zweiten Teilsimulation des Zeitschritts n verwendet. Die Ergebnisse der zweiten Teilsimulation werden als Eingangsgrößen der ersten Teilsimulation des Zeitschritts n+1 verwendet [131].

Prinzipiell ist in einer Co-Simulation nicht ein Eintreten von Teilsimulationen zur Laufzeit vorgesehen. Da dies durch Anforderung A4 gefordert wird, wird eine Co-Simulation, die dies ermöglicht, benötigt, im folgenden dynamische Co-Simulation bzw. „Plug-and-Simulate“-fähige Co-Simulation genannt.

2.4.2 Co-Simulation mit domänenspezifischen Standards

Hauptsächlich die Prozessindustrie und die Energiebranche verwenden schon seit langem Co-Simulationen und haben mehrere Co-Simulationsstandards und -ansätze hervorgebracht.

CAPE-OPEN (Computer-Aided Process Engineering) ist ein Standard der in der Prozessindustrie eingesetzt wird und Co-Simulation ermöglicht [132]. Allerdings spezialisiert sich CAPE-OPEN auf die Prozessindustrie und wird daher nicht in anderen Domänen eingesetzt [133]. Ein weiterer Nachteil von CAPE-OPEN ist, wie auch bei anderen Standards, dass es nicht die Möglichkeit bietet Simulationstools zur Laufzeit zu koppeln [133]. Diese Kopplung zur Laufzeit wird allerdings zur Realisierung von „Plug-and-Simulate“ benötigt.

Es gibt neben CAPE-OPEN noch eine Vielzahl an domänenspezifische Standards und Ansätze zur Co-Simulation. Hierunter fallen beispielsweise EPOCHS [134], Mosaik [135] und ADEVS [136], welche zur Simulation von Energienetzwerken und der damit verbundenen Kommunikation eingesetzt werden.

Diese weiteren domänenspezifischen Ansätze werden nicht weiterverfolgt, da sie nicht die Anforderung A3 eines domänenübergreifenden Einsatzes des Simulationskonzepts erlauben.

2.4.3 Dynamische Co-Simulation mit Functional Mock-up Interface

Das Functional Mock-up Interface (FMI) ist ein von der Automobilindustrie eingeführter Standard zur Co-Simulation und bietet zudem die Möglichkeit Modelle zwischen verschiedenen Simulationstools auszutauschen [137]. Allerdings müssen die Simulationstools FMI unterstützen [138]. Somit ist es nicht möglich Simulationstools in die Co-Simulation einzubinden, die FMI nicht unterstützen. Auch ist die Kommunikation zwischen den einzelnen Simulationstools auf diskrete Zeitpunkte beschränkt. Zwischen diesen Zeitpunkten werden die einzelnen Simulationen unabhängig voneinander gelöst. Ein Masteralgorithmus koordiniert hierbei den Datenaustausch zwischen den Subsystemen und die Synchronisation der Slave-Simulationen. FMI erlaubt zudem variable Größen der Zeitschritte zwischen den Synchronisationen. Bei einer Co-Simulation mit

FMI werden die einzelnen Subsimulationen durch sogenannte Functional Mock-up Units (FMU) vertreten, welche die Schnittstelle zu anderen Subsimulationen umsetzen [137]. Somit kann eine FMU als Blackbox gesehen werden.

Es gibt viele Beispiele in der Literatur, in denen FMI zur Co-Simulation genutzt wurde. Ein Beispiel für die Umsetzung einer Co-Simulation mit FMI wird in [139] gegeben. Zudem werden in [139] die Möglichkeiten von FMI erörtert. Hierbei hat sich gezeigt, dass der Masteralgorithmus vor Simulationsbeginn Informationen über die Slave-Simulationen benötigt, wodurch eine dynamische Co-Simulation nicht möglich ist.

2.4.4 Dynamische Co-Simulation mit High Level Architecture

High Level Architecture (HLA) ist eine vom amerikanischen Verteidigungsministerium entwickelt Architektur zur verteilten und parallelen Simulation [140]. Bei der HLA wird die gesamte Co-Simulation als Föderation bezeichnet. Eine Föderation besteht aus mehreren Föderaten, den einzelnen Subsimulationen, und einer zentralen Einheit zur Koordination der Föderaten, der Run-Time-Infrastructure (RTI). Durch die Interface-Specification werden die Schnittstellen zwischen den Föderaten und der RTI definiert. Das Object-Model-Template spezifiziert die Informationen die zwischen den einzelnen Föderaten ausgetauscht werden. Des Weiteren existieren die HLA-Regeln, die ein Simulator einhalten muss, um HLA-konform zu sein. Die RTI kann als Simulationsmaster angesehen werden und ist zusätzlich für die Synchronisation zwischen den einzelnen Föderaten verantwortlich [141]. HLA ermöglicht zwar den Eintritt von Komponenten zur Laufzeit [133] allerdings muss für jede Simulation ein neues Federation Agreement geschrieben werden, welches domänen- und anwendungsfallspezifisch ist [142]. Es gibt mehrere nutzbare Implementierungen einer RTI, sowohl frei zugängliche als auch kommerzielle Versionen [142].

In [143] wurden verschiedene Cyber-Physische Systeme mithilfe von mehreren Ptomely-II-Modellen simuliert. Ptomely-II ist ein Open-Source-Modellierungstool für heterogene Systeme. Die Co-Simulation wurde durch HLA realisiert. Allerdings stammen in diesem Beispiel die CPS alle aus einer Domäne, in diesem Fall aus der Luftfahrt. In [144] hingegen wurde ein einzelnes CPS mithilfe einer Co-Simulation simuliert, indem die Steuerungssoftware getrennt von den physikalischen Vorgängen simuliert wurden. [145] nutzt HLA um in der Cloud eine hochskalierbare Co-Simulation von IoT-Systemen zu ermöglichen. Somit ist mit HLA sowohl eine Simulation von vernetzten Komponenten, bei der jede Komponente individuell modelliert wird, als auch eine Simulation einer einzelnen Komponente mit verschiedenen Aspekten möglich.

2.4.5 Allgemeine Co-Simulationsansätze aus der Literatur

Zusätzlich gibt es in der Literatur wissenschaftliche Ansätze zur domänenspezifischen Co-Simulation von Internet-der-Dinge-Systeme. Hierunter fällt beispielsweise [146], wo Smart Grids

co-simuliert werden, indem in einer Simulation das Energienetz und in einer anderen Simulation die Kommunikation simuliert wird. Hier kommen allerdings nur zwei Simulationstools zum Einsatz deren Kopplung stark auf diese beiden abgestimmt ist. Auch [147] schlägt eine Co-Simulation für Smart Grids vor mit einer Trennung von Energienetz und Kommunikation. Es wird zwar kein Ansatz geliefert, allerdings wird eine Reihe von Co-Simulationsansätzen vorgeschlagen, die dies ermöglichen. Diese Ansätze, wie beispielsweise auch [148] werden aber in dieser Arbeit nicht weiter untersucht, da sie nur eine Kopplung von zwei Simulationstools vorsehen, was Anforderung A3 widerspricht. Auch Anforderung A4 wird nicht erfüllt, da ein Einfügen von Teilsimulationen zur Laufzeit nicht vorgesehen ist.

[149] bietet zwar eine weitfassendere Co-Simulationsarchitektur, allerdings stellt diese einen Ansatz dar, der nur bestimmte Anwendungsszenarien abdeckt und ein Einfügen von Teilsimulationen zur Laufzeit ist ebenfalls nicht vorgesehen, was Anforderung A4 nicht erfüllt.

2.4.6 Dynamische Co-Simulation mit SOA

Co-Simulation kann ebenfalls mit SOA (Service-Oriented Architecture) umgesetzt werden. Hierbei ist OPC UA (Open Platform Communications Unified Architecture) eine Möglichkeit dies zu realisieren.

OPC UA ist ein Machine-to-Machine Kommunikationsstandard der von der OPC-Foundation entwickelt wurde. OPC UA ist serviceorientiert und ermöglicht neben der Übertragung von Prozessdaten auch deren maschinenlesbare Beschreibung [150].

[151] setzt eine Co-Simulation mit Hilfe von OPC UA um. Hierbei wird jeder Simulator durch ein spezifisches Interface mit einem generischen Adapter verbunden, welcher einen OPC-UA-Server und einen OPC-UA-Client enthält. Diese Adapter kommunizieren mittels OPC UA mit einem zentralen Server welcher ebenfalls aus einem OPC-UA-Server und einem OPC-UA-Client besteht. Jeder Simulator muss sich am zentralen Server anmelden. Der erste Simulator der sich anmeldet ist der Simulations-Master, alle nachfolgenden Simulatoren sind Slaves. Der Master ist für die Koordination und die Synchronisation zwischen den einzelnen Simulatoren verantwortlich. Falls der Master aus der Simulation austritt, kann ein anderer Simulator Master werden.

Auch in [152] wird OPC UA zur Co-Simulation verwendet, indem über OPC UA eine Kommunikation und Synchronisation zwischen den Simulationstools Apros und OpenModelica ermöglicht wird. Dieser Ansatz ist auch auf weitere Simulationstools erweiterbar.

2.4.7 Dynamische Co-Simulation mit OSGi

OSGi ist ein Framework der Open-Services-Gateway-initiative, welches ein dynamisches Komponentensystem ermöglicht. OSGi basiert auf Java und wurde zur Entwicklung von

Anwendungen entwickelt, welche aus dynamisch zusammenstellbaren und wiederverwendbaren Komponenten bestehen. Bei OSGi wird die Implementierung einzelner Komponenten von den anderen Komponenten gekapselt. Die einzelnen Komponenten interagieren durch Services, wodurch sich die Komplexität des Systems verringert [153]. Die einzelnen Komponenten werden durch sogenannte Bundles repräsentiert, welche durch das Framework zur Laufzeit geladen, entfernt, ausgetauscht oder aktualisiert werden können. Diese Bundles können sich hierbei entweder auf einem oder verteilt auf mehreren Computern befinden [154].

Das dynamische Austauschen der Bundles ermöglicht die Realisierung einer dynamischen Co-Simulation wie sie in [155] umgesetzt wurde. Hier werden die einzelnen Simulationen durch jeweils ein Bundle repräsentiert. Zusätzlich werden Simulator-Coupler benötigt, um eine Verbindung zwischen den Simulationstools und den Bundles herzustellen und eine Synchronisation sowie einen Datenaustausch zu ermöglichen. Für diesen Simulator-Coupler wird OPC verwendet. Zusätzlich ist es aber auch möglich ein Modell direkt in ein OSGi-Bundle zu integrieren. Ein weiteres Bundle wird benötigt, um den Austausch von Simulatoren zur Laufzeit zu ermöglichen. In diesem Bundle werden die Zustände der entfernten Simulatoren gespeichert, um einen Wiedereintritt zu ermöglichen.

2.4.8 Dynamische Co-Simulation mit Agenten

Eine weitere Möglichkeit, die in der Literatur gegeben wird um eine Co-Simulation zu realisieren ist der Einsatz von Softwareagenten. Da Softwareagenten später eine bedeutende Rolle in der Realisierung des in Kapitel 3 vorgestellten Konzepts spielen, wird zuerst eine Einführung in Softwareagenten gegeben und anschließend wird die Literatur zur Co-Simulation mit Softwareagenten vorgestellt.

2.4.8.1 Eigenschaften von Software-Agenten

Es gibt mehrere Definitionen für Software-Agenten. Allen Definitionen ist gemein, dass ein Software-Agent eine gekapselte Softwareeinheit ist, die Autonomie und eine interoperable Schnittstelle besitzt und eine reale Einheit oder Software vertritt zur Verfolgung von deren Ziele [156]. Zusätzlich nehmen Software-Agenten ihre Umwelt wahr und interagieren mit ihr zum Erreichen dieser Ziele [157]. Weiterhin interagieren Software-Agenten mit anderen Software-Agenten, um ihre Ziele zu erreichen. Meistens ist diese Interaktion eine Kooperation zum Erreichen von gemeinsamen übergeordneten Zielen [158]. Im Weiteren werden Software-Agenten als Agenten bezeichnet.

Agentenarchitekturen lassen sich grundsätzlich in reaktive und deliberative Agenten einteilen [156].

In reaktiven Agenten hängt die Entscheidungsfindung direkt von der Situation ab, in der sich der Agent befindet. Jede Aktion eines Agenten ist eine Reaktion auf Daten, die vom Agenten erfasst

werden. Reaktive Agenten besitzen somit keinerlei komplexe Entscheidungsfindungs-algorithmen [156].

Deliberative Agenten besitzen ein internes Modell der Umwelt und versuchen durch Planung und Interaktion, Kooperation und Verhandlung mit anderen Agenten ihre Ziele zu erreichen [159]. BDI-Agenten (Belief, Desire, Intention) sind das bekannteste Beispiel für deliberative Agenten und wurden erstmals in [160] beschrieben.

Agieren mehrere Agenten zusammen zum Erreichen ihrer und gegebenenfalls gemeinsamer übergeordneter Ziele, so spricht man von einem Agentensystem.

Ein Agentensystem kann sich über mehrere Agentenplattformen erstrecken. Jede Agentenplattform besteht aus Agenten, einem Agenten-Management-System, einem oder mehreren optionalen Directory Facilitators und einem Nachrichten-Transport-System [161].

Die Agentenplattform stellt die physikalische Infrastruktur bereit, auf der Agenten laufen, wie eine Rechenmaschine, das dazugehörige Betriebssystem und gegebenenfalls Unterstützungssoftware für die Agenten. Agenten sind in der Lage mit anderen Agenten derselben Agentenplattform und mit Agenten außerhalb der Agentenplattform zu kommunizieren.

Der Directory Facilitator ist ein optionaler Bestandteil eines Agentensystems, in dem die Gelben Seiten des Agentensystems zur Verfügung gestellt werden. Bei den Gelben Seiten können die Agenten ihre eigenen Dienste anmelden und die Dienste anderer Agenten erfahren. Es können auch mehrere Directory Facilitators in einem Agentensystem existieren.

Das Agenten-Management-System ist ein zwingend notwendiger Teil eines Agentensystems. Das Agenten-Management-System verwaltet die Agentenplattform und die Agenten-IDs. Die Agenten-IDs sind in den Weißen Seiten hinterlegt, bei denen sich jeder Agent bei Eintritt in die Agentenplattform anmeldet. Durch die weißen Seiten können Agenten andere Agenten finden und eine Kommunikation mit ihnen aufbauen.

Über das Nachrichten-Transport-System können die einzelnen Agenten einer Agentenplattform miteinander kommunizieren. Die Kommunikation zwischen den Agenten wird durch die FIPA (Foundation for Intelligent Physical Agents) Agent Communication Language umgesetzt, welche in [162] spezifiziert ist.

Software bezeichnet jegliche zusätzliche Software auf die ein einzelner Agent Zugriff hat. In dieser können sich beispielsweise zusätzliche Anweisungen zum Verhalten des Agenten befinden.

2.4.8.2 Agentenbasierte Co-Simulationsansätze

In Kapitel 2.3.2 wurden einige Ansätze zur agentenbasierten Simulation vorgestellt, allerdings bietet keiner der vorgestellten Ansätze die Möglichkeit einer Co-Simulation mit beliebigen

Simulationstools. Allerdings findet man in der Literatur kaum Ansätze welche eine agentenbasierte Co-Simulation umsetzen.

Nur [30] verwendet Agenten zur Koordination einer multiphysikalischen Co-Simulation. Hierbei unterteilt ein Koordinationsagent das multiphysikalische Problem in physikalische Teilprobleme die dann von Berechnungsagenten, die verschiedene Simulationstools kapseln, gelöst werden. Jedes dieser Simulationstools wird hierbei von einem Agenten vertreten und somit werden die einzelnen Teilsimulationen über das Agentensystem gekoppelt. Allerdings ist auch in diesem Ansatz nicht das Einbinden von neuen Simulationstools zur Laufzeit vorgesehen.

2.4.9 Bewertung der Co-Simulationsansätze

Im Folgenden wird überprüft, welche der vorgestellten Ansätze die in Kapitel 1.3 genannten Anforderungen erfüllen. Anschließend wird diese Bewertungen diskutiert und eine Forschungslücke abgeleitet.

Anforderung A1: Modulare Modellierung

Da es sich bei den beschriebenen Ansätzen um Co-Simulationen handelt, bieten alle die Möglichkeit die Gesamtsimulation modular aufzubauen. Daher wird diese Anforderung von allen vorgestellten Ansätzen erfüllt.

Anforderung A2: Integration ohne Anpassung der Interaktionen

Auch wenn manche der vorgestellten Ansätze es prinzipiell technisch erlauben zur Laufzeit neue, zur Entwurfszeit unbekannte Modelle einzufügen, müsste in jedem Ansatz die für die Interaktionen mit anderen Modellen erforderlichen Verbindungen manuell erstellt werden. Dies wird genauer in der Diskussion zu Anforderung 4 erläutert. Die domänenspezifischen Ansätze erlauben oftmals schon kein Einfügen zur Laufzeit und wenn sie es erlauben, müssen hier jedes Mal die erforderlichen Verbindungen manuell hinzugefügt werden. FMI erlaubt ebenfalls kein Einbinden zur Laufzeit und neue Teilsimulationen müssen der Mastersimulation ebenfalls mitgeteilt werden. In HLA ist zwar prinzipiell das Einbinden von neuen Simulationen zur Laufzeit möglich, allerdings muss hierzu das Federation Agreement manuell angepasst werden. Bei den drei Ansätzen mit SOA, OSGi und Agenten ist ebenfalls ein Eintreten zur Laufzeit möglich, allerdings sehen auch hier die vorgestellten Ansätze kein Einbinden von zur Laufzeit unbekannten Simulationen vor, weshalb hier ebenfalls die Interaktionen manuell zugewiesen werden müssen. Keiner der Ansätze sieht grundlegend ein Einfügen oder Entfernen von zur Entwurfszeit unbekannten Modellen vor. Auch bei den weiteren allgemeinen Ansätzen in der Literatur ist eine Integration ohne Anpassung der Interaktionen nicht vorgesehen. Daher wird diese Anforderung von keinem der vorgestellten Ansätze erfüllt.

Anforderung A3: Heterogenität der Simulationstools

Die domänenspezifischen Co-Simulationen erfüllen aufgrund ihrer Domänenbeschränkung diese Anforderung nicht. FMI erfüllt diese teilweise, da sich die Anwendung mittlerweile über die Automobilbranche hinweg erweitert hat. Allerdings ist FMI immer noch auf die Produktion beschränkt, weshalb keine Simulationstools aus beispielsweise der Logistik oder der Energieversorgung eingebunden werden können. HLA ist sehr stark auf die Verteidigungsindustrie beschränkt. Somit können sehr viele Industrietools nicht eingesetzt werden und die Anforderung wird auch hier nicht erfüllt. Bei den drei Co-Simulationen mit SOA, OSGi und Agenten ist es zwar prinzipiell möglich, unterschiedliche Simulationstools anzubinden, allerdings ist der Fokus in den vorgestellten Ansätzen sehr stark auf einen bestimmten Anwendungsfall gerichtet, weshalb eine einfache Erweiterbarkeit um zusätzliche Simulationstools nicht gewährleistet ist und diese Anforderung daher nur im weitesten Sinne erfüllt wird. Bei den weiteren allgemeinen Ansätzen in der Literatur werden meist nur bestimmte Simulationstools verwendet, die auf den Anwendungsfall zugeschnitten sind und auf die dann die Co-Simulationsumgebung ebenfalls zugeschnitten ist, weshalb eine Erweiterbarkeit nicht einfach möglich ist. Somit wird diese Anforderung von keinem der vorgestellten Ansätze vollständig erfüllt.

Anforderung A4: Eintritt zur Laufzeit

Die meisten der domänenspezifischen Ansätze erlauben kein Eintreten von Teilsimulationen zur Laufzeit. Einige ermöglichen das Eintreten zur Laufzeit, jedoch ist eine manuelle Anpassung der Verbindungen zu den anderen Simulationstools und somit auch ein Stoppen der Gesamtsimulation notwendig. Daher wird diese Anforderung nur als teilweise erfüllt gewertet, da hierbei ein Stoppen der Gesamtsimulation notwendig ist. FMI hingegen lässt keinen Eintritt zur Laufzeit zu und erfüllt somit diese Anforderung nicht. HLA lässt zwar prinzipiell neue Teilsimulationen zur Laufzeit eintreten, allerdings muss es ebenfalls gestoppt werden um das Federation Agreement anzupassen, weshalb diese Anforderung ebenfalls nur teilweise erfüllt wird. Die restlichen drei Ansätze lassen prinzipiell problemlos einen Eintritt von neuen Simulationen zu, allerdings müssen hier ebenfalls die Verbindungen manuell angepasst werden, weshalb ebenfalls ein Stoppen bzw. Pausieren der Gesamtsimulation notwendig ist. Dies gilt allerdings nur für zur Entwurfszeit unbekannte Teilsimulationen, sind die Teilsimulationen bereits bekannt und die Verbindungen vordefiniert, können diese Teilsimulationen ohne Pausieren eingefügt werden. Die weiteren allgemeinen Ansätze in der Literatur sehen alle keinen Eintritt von Teilsimulationen zur Laufzeit vor. Tabelle 4 fasst diese Erkenntnisse zusammen.

Tabelle 4: Vergleich der Co-Simulationsansätze

Anforderung / Ansatz	A1	A2	A3	A4
Domänenspezifisch Co-Simulation	●	○	0	◑
Functional Mock-up Interface (FMI)	●	○	◑	○
High Level Architecture (HLA)	●	○	○	◑
Service-orientiert Architekturbasiert	●	○	◕	◕
OSGi-basiert	●	○	◕	◕
Agentenbasiert	●	○	◕	◕
Allgemeine Ansätze in der Literatur	●	○	◑	○

● Erfüllt ◕ Weitestgehend Erfüllt ◑ Teilweise Erfüllt ○ Nicht Erfüllt

Wie aus dieser Aufstellung ersichtlich wird, erfüllt keiner der existierenden Ansätze oder Standards alle Anforderungen an eine dynamische Co-Simulation, welche „Plug-and-Simulate“ ermöglicht. Anforderung A2, die Integration ohne Anpassung der Interaktionen, wird von keinem der gefundenen Ansätze erfüllt und auch Anforderung A3 (Heterogenität der Simulationstools) sowie Anforderung A4 (Eintritt zur Laufzeit) werden nie vollständig erfüllt.

2.5 Zusammenfassende Bewertung der Ansätze und Folgerung

Die vorgestellten Ansätze, Konzepte und Standards bieten alle interessante Lösungen für die Simulation von dynamischen IoT-Systemen. Zum einen gibt es einzelnen Simulatoren aus der Literatur, die spezifische Aspekte, wie Kommunikation, Skalierbarkeit, etc. adressieren. Diese ermöglichen es allerdings alle nicht unterschiedliche Simulationstools für unterschiedliche IoT-Komponenten zu verwenden. Dies wird aber, wie in Kapitel 1.2 hergeleitet, zunehmend an Bedeutung gewinnen, auch im Hinblick auf den Digitalen Zwilling, gerade wenn dieser herstellerübergreifend eingesetzt werden soll.

Dieses Problem der unterschiedlichen Simulationstools wird zwar von Co-Simulationsstandards und –ansätzen berücksichtigt, allerdings wird bei all diesen Standards und Ansätzen die Interaktionen zwischen den einzelnen Teilsimulationen zentral verwaltet, was ein nahtloses, dynamisches Einbinden von neuen Teilsimulationen zur Laufzeit nicht erlaubt. So werden diese Standards und Ansätze zwar der Heterogenität gerecht, allerdings nicht der geforderten Dynamik, welche ein nahtloses „Plug-and-Simulate“ zulässt.

Somit fehlt es an Ansätzen welche sowohl der Heterogenität als auch der Dynamik von IoT-Systemen gerecht werden. Co-Simulationen bieten zwar eine vielversprechende Grundlage, allerdings müssen die bisher bestehenden Standards und Ansätze um einen Ansatz, der die Möglichkeit eines nahtlosen „Plug-and-Simulate“ bietet, erweitert werden.

Daher ergibt sich folgende Forschungslücke, welche durch diese Arbeit geschlossen werden soll:

Es fehlt ein Ansatz, welcher es ermöglicht per „Plug-and-Simulate“ Teilsimulationen in eine Gesamtsimulation zur Laufzeit zu integrieren.

In diesem Kapitel wurde der Stand der Forschung und Technik bezüglich der Simulation von IoT-Systemen aufgezeigt und diskutiert. Zuerst wurde der aktuelle Stand der Technik Simulation allgemein vorgestellt und danach der aktuelle Stand der Technik des Internet der Dinge. Anschließend wurden bestehende Ansätze zur Simulation von IoT-Systemen präsentiert. Da diese aber nicht die Anforderungen die sich aus der Heterogenität ableiten erfüllen können, wurde zusätzlich noch der Stand der Forschung und Technik von dynamischen Co-Simulationen vorgestellt. Abschließend wurde durch einen Abgleich mit den in Kapitel 1.3 aufgestellten Anforderungen gezeigt, dass keiner der existierenden Ansätze alle Anforderungen erfüllt und eine Forschungslücke existiert. Diese Forschungslücke soll mit dem Konzept, welches im nächsten Kapitel vorgestellt wird geschlossen werden.

3 Konzept zur dynamischen Co-Simulation

Ausgehend von der in Kapitel 1 gestellten Forschungsfrage, wie per „Plug-and-Simulate“ Teilsimulationen in eine Gesamtsimulation zur Laufzeit integriert werden können und der in Kapitel 2 erlangten Erkenntnisse, dass solch eine nahtlose Integration durch existierende Co-Simulationsansätze nicht möglich ist, wird im Folgenden ein Konzept zu einer „Plug-and-Simulate“-fähigen Co-Simulation vorgestellt.

Trotz der Wahl eines Themas von praktischem Interesse ist die Praxisrelevanz wissenschaftlicher Arbeit oft unzureichend, so dass die wissenschaftlichen Erkenntnisse letztlich nicht in die Praxis umgesetzt werden können [163][164]. Zur Lösung dieses Problems wurde Design Science Research entwickelt, was eine anerkannte Methode zur Gewinnung von Erkenntnissen über ein Informationssystem [165][164] ist und einen iterativen Ansatz sowohl für die wissenschaftlichen Grundlagen als auch für die praktische Relevanz der Forschungsergebnisse [163] bietet. Design Science Research konzentriert sich auf die Erstellung und Bewertung von IT-Artefakten [163][166][167] und besitzt eine iterative Struktur, was sie flexibel macht und es ermöglicht kurzfristige Veränderungen sowie neue Erkenntnisse zu berücksichtigen. Die Grundidee ist, ein Artefakt zu entwickeln und dann das entwickelte Artefakt mit empirischen Daten zu evaluieren, um das Artefakt dann in einem neuen Zyklus zu verbessern. Ein Artefakt kann ein Konstrukt, ein Modell, eine Methode oder eine Instanz sein. Durch diese Iterationen wird eine ständige Überprüfung des Konzepts durch empirische Daten gewährleistet, was die Qualität und die praktische Relevanz des Konzepts verbessert [164]. Für diese Arbeit wurde im speziellen das Vorgehen nach Wieringa gewählt, dessen Modell sich aus den 5 Phasen „Problem Investigation“, „Treatment Design“, Treatment Validation“, „Treatment Implementation“ und „Implementation Evaluation“ zusammensetzt, durch die iteriert werden kann [168]. Dieses Modell wurde gewählt, da es sowohl eine Iteration über die Phasen „Problem Investigation“, „Treatment Design“, Treatment Validation“ sowie eine Iteration über alle Phasen zulässt, was eine einfache Validierung und Evaluierung der Ergebnisse anhand der Industriekooperation zuließ. Insbesondere die Iterationen über die ersten drei Phasen erwiesen sich hierbei als wertvoll und zielführend. Zudem ist das Modell nach Wieringa eines der gängigsten Modelle um Engineering-Probleme nach der Design Science Research zu lösen. Einige der nun vorgestellten Konzeptteile wurden bereits in [83], [169], [170], [171], [172] vorgestellt, eine umfassende Betrachtung steht allerdings noch aus.

Hierzu wird zunächst in Unterkapitel 3.1 eine grundlegende Diskussion geführt, ob es sinnvoller ist, eine FMI-ähnliche Marktlösung, bei der die Hersteller die Simulationstools in die Co-Simulationsplattform integrieren müssen, zu favorisieren oder ob ein Konzept für eine Plattform entwickelt werden sollte, welche die Simulationstools unabhängig vom Hersteller integriert. Anschließend wird ein Überblick über das sich aus dieser Diskussion ergebende Konzept für die Plattform und für die Integration der Simulationstools gegeben sowie eine Beschreibung der Co-

Simulationsumgebung und der dazugehörenden Simulationsvertreter. Anschließend werden Einschränkungen an die Teilsimulationen aufgezeigt, welche definieren, welche Bestandteile der Komponenten modelliert werden müssen, um eine „Plug-and-Simulate“-fähige Co-Simulation zu ermöglichen.

In Unterkapitel 3.2 wird der Datenaustausch in der Co-Simulation erläutert. Hierzu wird zunächst ein Überblick über den Datenaustausch in existierenden Co-Simulationsstandards gegeben und anschließend werden diese Datenaustauschansätze bewertet. Ausgehend von dieser Bewertung wird ein Konzept vorgestellt, welches den Datenaustausch in einer dynamischen Co-Simulation ermöglicht. Abschließend wird in diesem Unterkapitel erläutert, welche Einschränkungen der Datenaustausch auf die Modellierung der Komponenten hat.

Anschließend werden in Kapitel 3.3 die für den Datenaustausch notwendigen Interaktionssimulationen vorgestellt, beginnend mit einer Einordnung von Interaktionsarten im IoT. Anschließend werden die Konzepte zur kommunikationsorientierten Interaktionssimulation und zur prozessorientierten Interaktionssimulation vorgestellt und wie die entsprechenden Schnittstellen konzipiert werden müssen. Abschließend wird wieder erläutert, welche Einschränkungen die Simulation der Kommunikation und prozessorientierten Interaktion auf die Modellierung haben.

Zuletzt wird in Kapitel 3.4 die Synchronisation der Co-Simulation vorgestellt. Da es nicht Ziel dieser Arbeit ist, eine neue Synchronisation zu entwickeln, werden zunächst bestehende Synchronisationsansätze untersucht und bewertet. Ausgehend von dieser Bewertung wird das gewählte Konzept zur Synchronisation der Co-Simulation, bestehend aus Taktgeber und Synchronisationsschnittstelle beschrieben. Abschließend wird wieder erläutert, welche Einschränkungen die Synchronisation auf die Modellierung hat.

Ein ausgewiesenes Ziel dieser Arbeit ist es, eine nahtlose Einbindung von bereits existierenden Teilsimulationen in eine Co-Simulation zu ermöglichen. Hierbei ist ebenfalls gefordert, dass es möglich sein soll, auch Teilsimulationen von den Komponentenherstellern zu verwenden, also Teilsimulationen, die nicht selbst entwickelt wurden. Zu dieser Forderung stehen die in jedem Unterkapitel enthaltenen Einschränkungen für die Modellierung der Teilsimulationen. Diese Einschränkungen sind allerdings derart, dass gefordert wird, dass die Modelle grundsätzlich Co-Simulationsfähig sind, beispielsweise, dass sie Schnittstellen nach außen enthalten und für eine Synchronisation pausierbar sein müssen. Somit ist dies kein Widerspruch zu dem Ziel Teilsimulationen von Komponentenherstellern nahtlos einbinden zu können, da ohne diese Einschränkungen eine Co-Simulation nicht möglich wäre. Dies wird in den Diskussionen der Einschränkungen in den einzelnen Unterkapiteln deutlich.

3.1 Grundlegende Entscheidungen und Konzeptübersicht

3.1.1 Entscheidung zur Einbindung von Simulationstools

Grundwesen einer Co-Simulation ist, dass Teilsimulationen in meist unterschiedlichen Simulationstools eine Gesamtsimulation ausführen. Aufgabe einer Co-Simulationsumgebung ist dies zu ermöglichen, also eine Verknüpfung der Simulationstools herzustellen. Um solch eine Verknüpfung zu ermöglichen, werden Schnittstellen zu den einzelnen Simulationstools benötigt. Zur Realisierung dieser Schnittstellen gibt es aus Sicht einer Co-Simulationsumgebung zwei grundlegende Möglichkeiten:

(1) Standardorientierter Ansatz: Die erste Möglichkeit folgt dem Ansatz von Standards, wie es beispielsweise bei Functional Mock-up Interface (FMI) der Fall ist. Hierzu definiert die Co-Simulationsumgebung einen Standard, wie die Schnittstellen, welche die Simulationstools bereitstellen müssen, auszusehen haben und die Simulationstools müssen dann diese Schnittstellen umsetzen.

(2) Open-Source-orientierter Ansatz: Bei der zweiten Möglichkeit, implementiert die Co-Simulationsumgebung selbst die Schnittstellen zu den einzelnen Simulationstools. Hierzu greift sie auf die bereits an den Simulationstools existierenden und zugänglichen Schnittstellen zu und greift darüber Daten ab bzw. speist Daten in die Teilsimulation ein. Der für die Implementierung der Schnittstellenzugriffe nötige Quellcode wird hierbei offengelegt, wodurch es somit jedem möglich ist, ein Simulationstool an die Co-Simulationsumgebung anzubinden.

Im Folgenden soll diskutiert werden, welche Vor- und Nachteile diese beiden Ansätze mit sich bringen und welcher Ansatz besser für eine „Plug-and-Simulate“-fähige Co-Simulation geeignet ist. Abhängig von dieser Diskussion müssen die Schnittstellen unterschiedlich im folgenden Co-Simulationskonzept berücksichtigt werden.

Vorteile des standardorientierten Ansatzes / Nachteile des Open-Source-orientierten Ansatzes:

- Ein wesentlicher Vorteil des standardorientierteren Ansatzes ist, dass in der Definition der Standardisierung der Schnittstellen die vorhin erwähnten Einschränkungen an die Simulationstools festgelegt werden können. Somit ist beispielsweise gewährleistet, dass ein Simulationstool die gewünschte Synchronisation bereitstellt und die Daten im gewünschten Format vorliegen, ohne dass die Co-Simulationsumgebung darauf reagieren muss. Beim Open-Source-orientierten Ansatz hingegen können diese Einschränkungen nicht direkt an die Simulationstools weitergegeben werden, sondern müssen entweder bei der Modellierung der Komponenten berücksichtigt werden oder von der Co-Simulationsumgebung aufgefangen werden. Somit müssen bei der Umsetzung der

Schnittstellen eventuell Workarounds implementiert werden, welche die Performanz der Co-Simulation verringern, um ein Simulationstool anbinden zu können.

- Ein weiterer Vorteil des standardorientierterteren Ansatzes ist, dass wenn die Simulationstoolhersteller die Schnittstellen implementieren eine industrielle Implementierung vorliegt, welche ausgiebig getestet wurde und somit meist stabiler als eine Open-Source-Software läuft. Beim Open-Source-orientierten Ansatz hingegen kennen die Implementierer der Schnittstelle oftmals das Simulationstool nicht so genau und können somit Eigenheiten des Simulationstools nicht so gut abfangen, weshalb die Schnittstelle fehleranfälliger werden kann.

Nachteile des standardorientierten Ansatzes / Vorteile des Open-Source-orientierten Ansatzes:

- Ein Nachteil des standardorientierterteren Ansatzes ist, dass man auf die Implementierung der Hersteller angewiesen ist und somit auf die Simulationstools die diese Schnittstellen implementiert haben limitiert ist. Meistens liegt der Quellcode des Simulationstools nicht offen, weshalb nur der Hersteller selbst die gewünschten Schnittstellen implementieren kann. So ist es nicht möglich selbst ein für die Co-Simulation benötigtes Simulationstool zu integrieren, was die Anzahl der potentiellen Simulationstools und somit die Heterogenität limitiert. Beim Open-Source-orientierten Ansatz hingegen können potentiell alle Simulationstools, welche eine Co-Simulation prinzipiell ermöglichen, an die Co-Simulation angebunden werden, wodurch ein deutlich größerer Pool an nutzbaren Simulationstools entsteht, welcher die Heterogenität eines IoT-Systems besser widerspiegelt. Diese Integration eines neuen Simulationstools kann sowohl vom Co-Simulationsbetreiber, als auch vom Co-Simulationsnutzer und vom Simulationstoolhersteller selbst vorgenommen werden.

- Ein weiterer Nachteil des standardorientierterteren Ansatzes ist, dass ein Anreiz für den Simulationstoolhersteller existieren muss, damit ein Simulationstoolhersteller sein Simulationstool in die Co-Simulationsumgebung integriert, welcher nur bei einer ausreichenden Marktmacht vorhanden ist. Bei FMI beispielsweise war dies möglich, da hier die Automobilhersteller gemeinsam diesen Standard definiert hatten und somit einen Anreiz für die Zulieferer schufen. Beim Open-Source-orientierten Ansatz wird eine solche Marktmacht vom Co-Simulationsumgebungsbetreiber nicht benötigt, da er selbst die Anbindung an die Simulationstools realisieren und anbieten kann.

Ein Vergleich der Vor- und Nachteile der beiden Ansätze zeigt, dass eine Co-Simulationsumgebung nach dem standardorientierten Ansatz stabiler und performanter laufen würde, allerdings wird für diese Lösung eine Marktmacht benötigt, die die Simulationstoolhersteller veranlasst, ihre Simulationstools einzubinden. Zudem ist die Anzahl der

potenziell nutzbaren Simulationstools auf die durch die Hersteller eingebunden Simulationstools begrenzt.

Aus wissenschaftlicher Sicht ist es wichtig, eine Vielzahl an unterschiedlichen Simulationstools einzubinden (Mehrwert 2 aus Kapitel 1.3) und nicht eine möglichst performante Co-Simulationsumgebung zu erstellen. Daher wurde der Open-Source-orientierten Ansatz ausgewählt.

Aufgrund dieser grundlegenden konzeptionellen Entscheidung muss das Konzept der reinen Co-Simulationsumgebung noch um ein Konzept zur Anbindung von Simulationstools an diese Co-Simulationsumgebung erweitert werden. Im Folgenden wird ein Überblick über das gesamte Co-Simulationskonzept gegeben.

3.1.2 Überblick über die Co-Simulation

In dieser Arbeit wird eine zur Laufzeit erweiterbare Co-Simulation vorgestellt, die sowohl die Kapselung der einzelnen Komponenten, deren Datenaustausch sowie die zeitliche Synchronisierung der einzelnen Teilsimulationen beinhaltet. Eine Co-Simulation ist eine Zusammensetzung aus mehreren Teilsimulationen, wobei die Modelle oftmals in unterschiedlichen Simulationstools ausgeführt werden, wie in Abbildung 3 dargestellt. Da Anforderung A1 eine Integration von neuen Teilmodellen ohne Anpassungsaufwand an anderen Teilmodellen fordert, werden die einzelnen IoT-Komponenten in separaten Teilsimulationen ausgeführt, im weiteren Komponentensimulationen genannt. Das heißt, jede IoT-Komponente wird für sich in einer eigenen Komponentensimulation simuliert (siehe Abbildung 3).

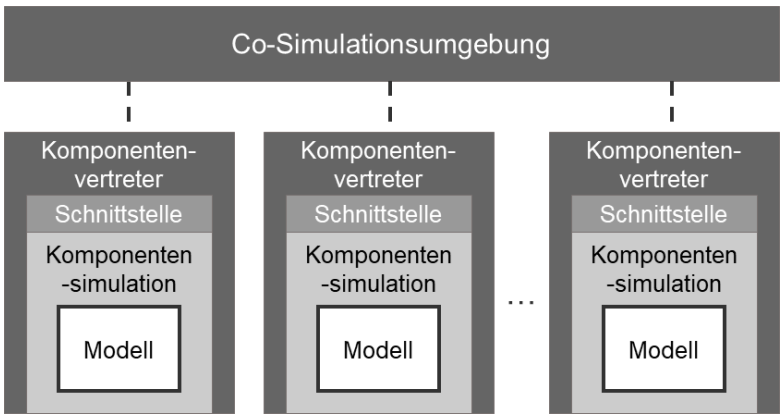


Abbildung 3: Co-Simulation mit Teilsimulationen

Aufgrund der in Kapitel 3.1.1 getroffenen Entscheidung, dass ein Open-Source-orientierter Ansatz zur Umsetzung der Co-Simulation gewählt werden soll, und aufgrund von Anforderung A3, dass unterschiedliche Simulationstools integrierbar sein müssen, müssen die Schnittstelle zwischen der Co-Simulationsumgebung und den einzelnen Simulationstools zusätzlich berücksichtigt werden. Dies wird in Abbildung 3 durch die an die Komponentensimulationen

anliegenden Schnittstellen verdeutlicht. Eine detaillierte Beschreibung der Schnittstellen wird in Kapitel 3.2 gegeben.

Anforderung A4 fordert eine Integration von neuen Teilsimulationen, und somit von neuen Komponentensimulationen zur Laufzeit. Daher ist es nicht möglich die Schnittstellen zwischen der Co-Simulationsumgebung und den Teilsimulationen fest in die Co-Simulationsumgebung zu integrieren, sondern es wird ein Simulationsvertreter (Komponentenvertreter) zwischengeschaltet, der die Co-Simulationsumgebung zur Laufzeit betreten und verlassen kann, wodurch Anforderung A4 berücksichtigt wird. Der Simulationsvertreter wird auch deshalb zwischengeschalten, um eine saubere Trennung zwischen den einzelnen Funktionen zu erhalten: die Schnittstelle ist für die Verbindung zum Simulationstool zuständig, der Simulationsvertreter für das dynamische Ein- und Austreten. Zusätzlich kapselt dieser Komponentenvertreter die Teilsimulationen gegenüber der Co-Simulationsumgebung wodurch Anforderung A1 ebenfalls erfüllt wird. Somit besitzt der Komponentenvertreter kein Wissen über den tatsächlichen Inhalt der Teilsimulation oder über den Inhalt der Daten, die diese austauscht. Über diese komplette Kapselung wird ein Eintritt zur Laufzeit möglich, da nur die einzutretende Teilsimulation mit ihrem Komponentenvertreter verbunden werden muss und sonst keine weiteren Anpassungen vorgenommen werden müssen. Diese Komponentenvertreter sind über die Co-Simulationsumgebung miteinander verbunden und können über sie miteinander interagieren. Somit sind auch die Komponentensimulationen über die Schnittstellen und die Komponentenvertreter über die Co-Simulationsumgebung miteinander verbunden und können miteinander interagieren.

Anforderung A2 fordert eine Integration von Teilsimulationen ohne manuellen Anpassungsaufwand der möglichen Interaktionen. Um dies zu ermöglichen muss eine direkte Interaktion zwischen den einzelnen Teilsimulationen vermieden werden, da diese bei einem Eintritt einer neuen Simulation für jede Teilsimulation angepasst werden müssten. Trotzdem müssen auch die Interaktionen zwischen den einzelnen Teilsimulationen modelliert und simuliert werden. Daher wird eine Interaktionssimulation eingefügt, welche die Interaktionen kapselt. Diese wird ebenfalls in einer eigenen Simulation ausgeführt und ist ebenfalls über eine Schnittstelle und einen Simulationsvertreter (Interaktionsvertreter) mit der Co-Simulationsumgebung verbunden (siehe Abbildung 4).

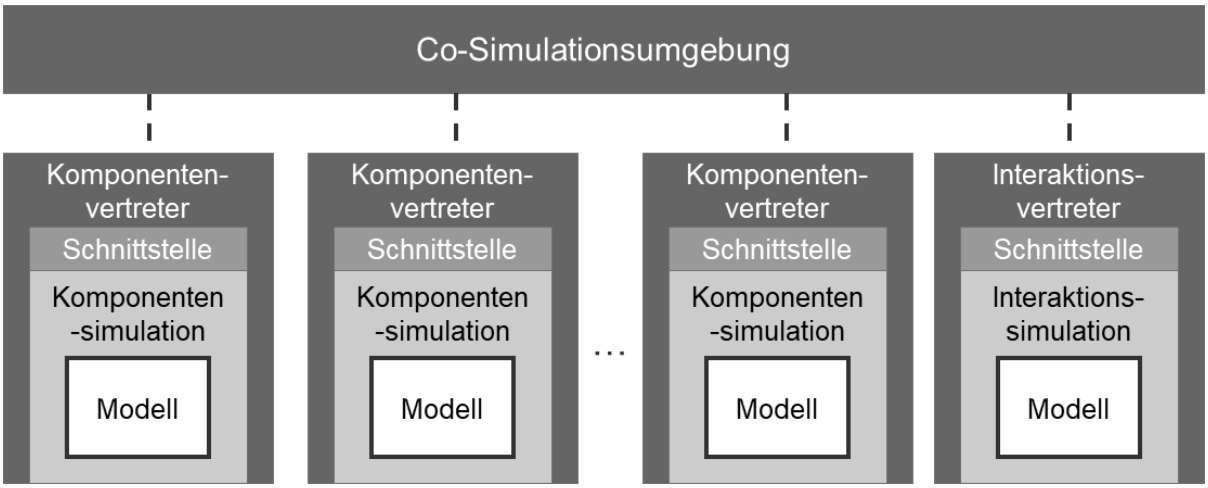


Abbildung 4: Co-Simulation mit Interaktionssimulation

Ist im Folgenden die Rede von Simulationsvertretern, so gilt die Aussage sowohl für Komponentenvertreter als auch für Interaktionsvertreter.

Alle Interaktionen zwischen den Komponentensimulationen laufen über diese Interaktionssimulation ab, das heißt, wenn Komponentensimulation A mit Komponentensimulation B interagieren möchte, sendet sie diese Interaktion über die Schnittstelle und den Komponentenvertreter an den Interaktionsvertreter der Interaktionssimulation welcher die Interaktion an die Interaktionssimulation weiterleitet. Diese simuliert dann die Interaktion und gibt sie an Komponentensimulation B weiter. Somit sind die einzelnen Komponentensimulationen voneinander gekapselt und wenn eine neue Komponentensimulation eintritt müssen keine Interaktionspfade zu anderen Komponentensimulationen gelegt werden und der Komponentenvertreter der neuen Komponentensimulation muss nicht wissen, an welche andere Komponentensimulation eine Interaktion gerichtet ist, sondern muss eine Interaktion nur an die Interaktionssimulation senden, welche sie durch das Simulieren der Interaktion an die richtige Komponentensimulation weiterleiten kann. Eine detaillierte Beschreibung der Interaktionssimulation und warum diese beim Eintritt einer neuen Komponentensimulation nicht angepasst werden muss, wird in Kapitel 3.3 gegeben.

Zusätzlich zum Datenaustausch, der die Interaktionen der Teilsimulationen in einer Co-Simulation ermöglicht, wird eine Synchronisation der Co-Simulation benötigt. Wenn die einzelnen Teilsimulationen unterschiedlich schnell laufen, was insbesondere dann auftritt, wenn diese in unterschiedlichen Simulationstools ausgeführt werden, kann es zu Kausalitätsfehlern kommen. Daher wird die in Abbildung 4 präsentierte Co-Simulation um einen Taktgeber erweitert, siehe Abbildung 5. Dieser koordiniert die einzelnen Teilsimulationen über die Co-Simulationsumgebung und die zu den Teilsimulationen gehörenden Komponentenvertreter indem er ein Pausieren der Simulationen zu gegebenen Zeitpunkten veranlasst. Eine detaillierte Beschreibung der Synchronisation wird in Kapitel 3.4 gegeben.

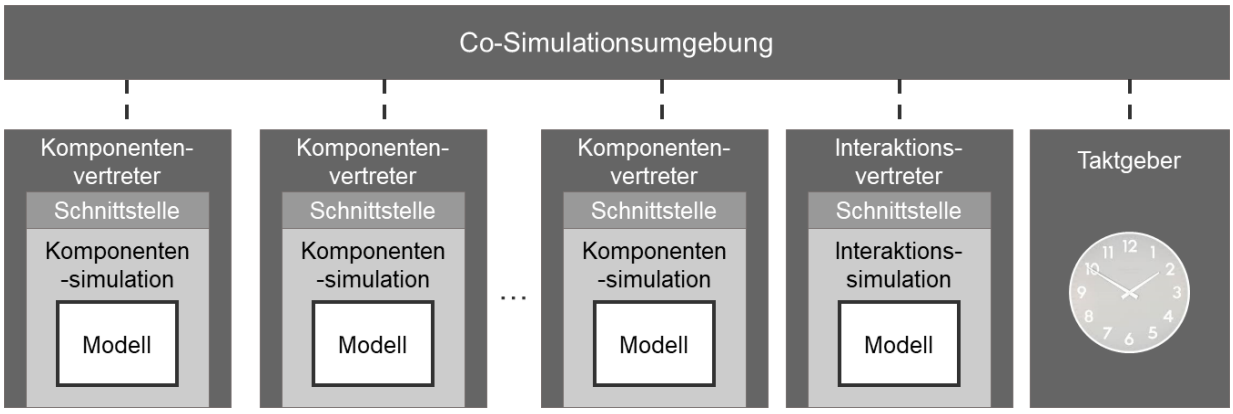


Abbildung 5: Co-Simulation mit Taktgeber

3.1.3 Co-Simulationsumgebung und Simulationsvertreter

3.1.3.1 Co-Simulationsumgebung

Die Hauptaufgabe der Co-Simulationsumgebung besteht darin, die Interaktionen, die durch die Komponentensimulationen über die Komponentenvertreter versendet werden, an die Interaktionssimulation über deren Interaktionsvertreter weiterzuleiten. Anschließend muss die Co-Simulationsumgebung die von der Interaktionssimulation verarbeiteten Interaktionen an die Zielkomponentensimulation bzw. deren Komponentenvertreter weiterleiten (siehe Abbildung 6: Nachrichtenweiterleitung). Die Interaktionsweiterleitung wird in den Kapiteln 3.2 und 3.3 detailliert beschrieben.

Zusätzlich ermöglicht die Co-Simulationsumgebung die durch den Taktgeber koordinierte Synchronisation. Hierzu muss sie die von ihm gesendeten bzw. von ihm zu empfangenden Synchronisationsnachrichten weiterleiten und an die entsprechenden Simulationsvertreter (Komponenten- und Interaktionsvertreter) zustellen. Hierzu kann ebenfalls auf die Nachrichtenweiterleitung zugegriffen werden (siehe Abbildung 6: Nachrichtenweiterleitung). Die Synchronisation wird in Kapitel 3.4 detailliert beschrieben.

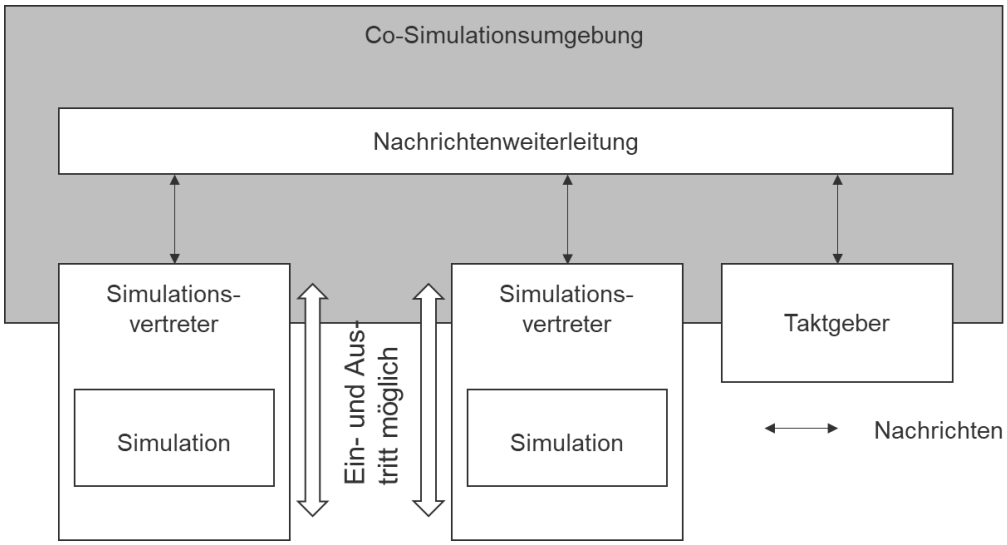


Abbildung 6: Aufbau der Co-Simulationsumgebung und der Simulationsvertreter

Zudem muss sie um eine dynamische, „Plug-and-Simulate“-fähige Co-Simulation zu ermöglichen (Anforderung A4), in die zur Laufzeit Teilsimulationen ein- und austreten können, eine Möglichkeit bieten, dass die Simulationsvertreter, die die Teilsimulationen kapseln, zur Laufzeit in die Co-Simulationsumgebung ein- und austreten (siehe Abbildung 6). Dies muss geschehen können, ohne dass die Co-Simulationsumgebung pausiert werden muss.

3.1.3.2 Simulationsvertreter

Die Simulationsvertreter (Komponenten- und Interaktionsvertreter) können als Teil der Co-Simulationsumgebung gesehen werden, müssen aber unabhängig von ihr existieren und agieren können. Sie müssen in der Lage sein, die Co-Simulationsumgebung zur Laufzeit zu betreten oder zu verlassen, ohne dabei selbst pausiert zu werden.

Zusätzlich können sie Interaktionen über Nachrichten an die Nachrichtenweiterleitung der Co-Simulation versenden und von dieser empfangen (siehe Abbildung 6: Nachrichten). Sie sind zudem in der Lage die vom Taktgeber versendeten Synchronisationsnachrichten an die Komponentensimulationen weiterzuleiten.

Um die Interaktionen richtig weiterleiten zu können, ist den Simulationsvertretern bekannt, ob sie eine Komponentensimulation oder eine Interaktionssimulation vertreten (siehe Abbildung 6: Simulationsart). Somit existiert ebenfalls eine Unterscheidung zwischen Komponentenvertreter und Interaktionsvertreter.

Das Konzept, das es ermöglicht, dass die Interaktionsvertreter der Interaktionssimulationen die Nachrichten an die richtigen Komponentensimulationen zu versenden, wird in Kapitel 3.3 detailliert beschrieben. Zudem werden die einzelnen Simulationsvertreter um weitere Bestandteile in den folgenden Kapiteln erweitert.

3.1.4 Anforderungen an die Teilsimulationen und ihre Simulationstools

Im oben beschriebenen Konzept wurde die Entscheidung getroffen, jede IoT-Komponente separat zu modellieren und zu simulieren. Zudem wurde die Entscheidung getroffen, dass die Co-Simulationsumgebung und die Simulationsvertreter selbst keine Modellierung oder Simulationen enthalten, sondern nur für die Verknüpfung der Teilsimulationen und den damit verbundenen Aufgaben zuständig sind. Hierdurch ergibt sich die Anforderung an die Teilsimulationen, in diesem Fall die Komponentensimulationen, dass in ihnen die Komponenten ganzheitlich modelliert und simuliert werden. Das heißt, alle für die Gesamtsimulation relevanten Aspekte müssen in den Komponentensimulationen modelliert und simuliert werden. Es ist somit nicht möglich, wie beispielsweise bei anderen Co-Simulationen Teilaspekte der Komponenten in andere Teilsimulationen auszulagern.

Sollte es nicht möglich sein, alle relevanten Teilaspekte einer Komponente in einer Simulation umzusetzen, so ist es prinzipiell möglich, eine Teilsimulation ebenfalls als Co-Simulation umzusetzen, welche dann durch den Simulationsvertreter der Co-Simulationsumgebung und somit den restlichen Teilsimulationen gegenüber als eine Simulation vertreten wird (siehe Abbildung 7). Synchronisation und Datenaustausch in dieser Komponenten Co-Simulation hängen von der gewählten Co-Simulationsumgebung ab.

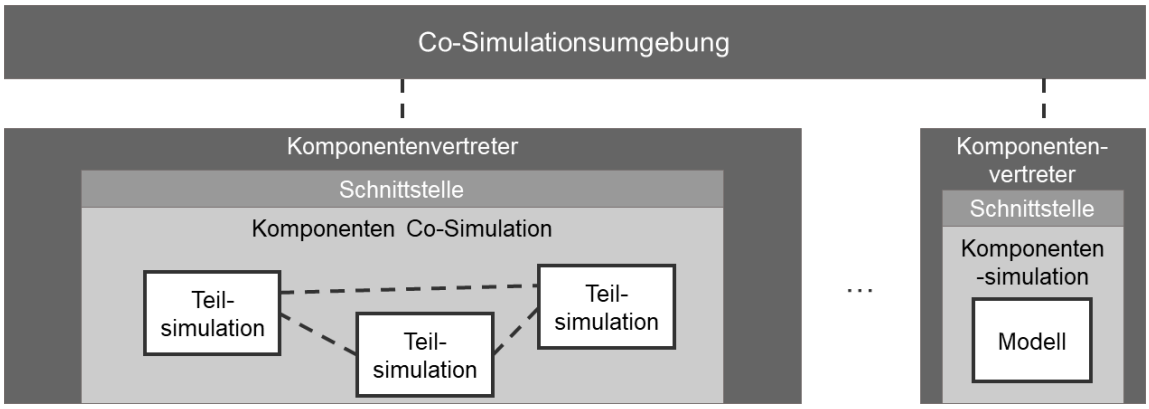


Abbildung 7: Komponentensimulation als Co-Simulation

Dieses Vertreten als eine Simulation ist nur eine Einschränkung für die Verteilung der Modellierung, durch die es nicht mehr möglich ist, eine Komponente zu modellieren, die nicht durch einen Komponentenvertreter gekapselt ist. Hätte man diese Einschränkung nicht, so wäre eine dynamische, „Plug-and-Simulate“-fähige Co-Simulation nicht realisierbar, da beim Einfügen einer neuen Komponentensimulation andere Teilmodelle angepasst werden müssten. Geht man allerdings wie in Kapitel 1.2 erläutert davon aus, dass Modelle der Komponentenhersteller wiederverwendet werden können, so ist dies keine Einschränkung, da in diesem Fall sowieso alle relevanten Aspekte der Komponente schon durch den Komponentenhersteller in einer Simulation umgesetzt wurden.

Darüber hinaus müssen die gewählten Simulationstools eine Synchronisation ermöglichen. Dies ist keine Einschränkung, sondern eine Voraussetzung für jede Co-Simulation, da ohne eine Synchronisation keine Co-Simulation stabil lauffähig ist. In Kapitel 3.4 wird eine mögliche Synchronisation vorgestellt.

3.2 Datenaustausch in Co-Simulationen

In diesem Unterkapitel wird das Konzept zum Datenaustausch der Interaktionen in der Co-Simulation beschrieben. Dieser unterteilt sich zum einen in den Datenaustausch zwischen den Simulationstools und den Simulationsvertretern, sowie den Datenaustausch zwischen den Simulationsvertretern über die Co-Simulationsumgebung welcher im Prinzip nur eine Weiterleitung des Datenaustauschs zwischen Teilsimulationen ermöglicht. Zum anderen unterteilt er sich auch in einen interaktionsbezogenen und einen synchronisationsbezogenen Datenaustausch, welcher ansatzweise schon in Kapitel 3.1.3 beschrieben wurde.

In diesem Kapitel wird hauptsächlich der interaktionsbezogene Datenaustausch beschrieben und nur eine Grundlage für die Synchronisation gelegt. Der für die Synchronisation benötigte Datenaustausch wird ausführlicher in Kapitel 3.4 beschrieben.

Zuerst wird der Datenaustausch in bereits bestehenden Co-Simulationsstandards (FMI und HLA) untersucht und auf eine Übertragbarkeit auf diese Arbeit hin bewertet. Anschließend wird der Datenaustausch zwischen den Simulationsvertretern über die Co-Simulationsumgebung erklärt, gefolgt von einem Konzept zur Anbindung von Simulationstools an die Simulationsvertreter über die bereits in Kapitel 3.1 erwähnten Schnittstellen. Abschließend werden die Anforderungen die sich durch dieses Konzept an die Teilsimulationen ergeben dargelegt.

3.2.1 Datenaustausch bei Co-Simulationsstandards

Ziel dieser Arbeit ist es ein neues Konzept, welches eine „Plug-and-Simulate“-fähige Co-Simulation ermöglicht, zu entwickeln, nicht ein neuartiges Datenaustauschkonzept für eine Co-Simulation. Daher werden bereits bestehende Datenaustauschkonzepte von den bereits existierenden Co-Simulationsstandards untersucht, um diese im Rahmen dieser Arbeit zu erweitern. Zuerst werden deshalb die Datenaustauschkonzepte von FMI und HLA, stellvertretend für Co-Simulationsstandards, untersucht und anschließend auf ihre Übertragbarkeit auf das in Kapitel 3.1 bereits vorgestellte Konzept hin untersucht.

Zusätzlich zu FMI und HLA werden noch Co-Simulationsansätze basierend auf OSGi [133] und OPC UA [151] untersucht, welche beide prinzipiell „Plug-and-Simulate“-fähig sind. Zu diesen Ansätzen gibt es allerdings keine Informationen, wie genau der Datenaustausch in der Co-Simulation abläuft.

3.2.1.1 Datenaustausch bei FMI

Bei FMI werden die einzelnen Simulatoren in Functional Mock-Up Units (FMUs) gekapselt und in ein Simulationstool integriert, welches als Simulationsmaster dient. Die in den FMUs gekapselten Simulatoren nehmen als Simulationsslaves an der Co-Simulation teil. Hierbei wird der Datenaustausch durch den Simulationsmaster zentral koordiniert, wodurch kein direkter Datenaustausch zwischen zwei Simulationsslaves möglich ist. Der Datenaustausch selbst findet immer zu diskreten Zeitpunkten statt, indem der Simulationsmaster C-Funktionen (Funktionen in der Programmiersprache C) der Simulationsslaves aufruft und ihnen so über Parameter die benötigten Daten liefert. Nach Beendigung des Simulationsschritts, übergeben die Simulationsslaves durch Rückgabewerte wieder Daten an den Simulationsmaster, welcher diese vor dem nächsten Simulationsschritt wieder an die entsprechenden Simulationsslaves verteilt [173].

3.2.1.2 Datenaustausch bei HLA

Der Datenaustausch wird auch bei HLA zentral koordiniert. Jede Co-Simulation bei HLA besitzt eine Runtime-Time-Infrastructure (RTI), welche viele Funktionen des Simulationsmasters in FMI erfüllt. Die RTI ist selbst allerdings keine Simulation, sondern hauptsächlich nur für den Datenaustausch und die Synchronisation der Co-Simulation zuständig. Der Datenaustausch selbst findet über den Austausch von Objekten statt und folgt dem Publish-Subscribe-Schema. Hierbei meldet jede Teilsimulation bei der RTI welche Daten sie bereitstellen kann und welche sie benötigt. Allerdings haben die Teilsimulationen keine Informationen über die anderen Teilsimulationen. Damit die RTI dennoch die Daten an die richtige Teilsimulation weiterleiten kann, existiert in jeder Co-Simulation mit HLA mindestens ein Federation Object Model, in welchem definiert ist, welche Daten in der Co-Simulation ausgetauscht werden können. Zusätzlich besitzt jede Teilsimulation ein Simulation Object Model, in dem spezifiziert ist, welche Daten sie bereitstellen kann und welche sie benötigt [174].

3.2.1.3 Bewertung der Ansätze

Prinzipiell bietet das Konzept des Datenaustauschs von FMI die Möglichkeit von „Plug-and-Simulate“, allerdings wird sie durch die zentrale Koordination des Datenaustausches durch den Simulationsmaster „Plug-and-Simulate“ erschwert, da das gesamte Wissen über die Co-Simulation an einem zentralen Ort aggregiert werden muss. Dies stellt eine Einschränkung bei der Modellierung und der Wahl der Simulationstools dar, da beim Eintritt einer neuen Simulation, diese dem Simulationsmaster mitteilen muss, welche Daten sie bereitstellen kann und welche sie benötigt. Dies ist durch die Kapselung der einzelnen Teilsimulationen durch die Simulationsvertreter nicht möglich, da die Simulationsvertreter kein Wissen über die tatsächlichen Daten haben, die von den Teilsimulationen versandt werden. Dies ist jedoch notwendig, da in Kapitel 3.1.1 der Open-Source-orientierte Ansatz gewählt wurde und somit es

nicht möglich ist, den Simulationsvertretern einheitlich Wissen über die Teilsimulationen zur Verfügung zu stellen ohne hohen manuellen Aufwand. Ein manueller Aufwand an der Gesamtsimulation steht jedoch im Widerspruch zu Anforderung A2. Daher wird eine komplette Kapselung durch die Simulationsvertreter gewählt und somit ist kein zentrales Datenaustauschkonzept, welches in der Co-Simulationsumgebung selbst verortet ist möglich.

Bei HLA können Modelle, die schon vor Simulationsstart bekannt sind und berücksichtigt wurden, zur Laufzeit der Simulation geladen werden. Zur Entwicklungszeit unbekannte Modelle können allerdings nicht zur Laufzeit geladen werden, ohne dass das Federation Object Model angepasst wird, was einen manuellen Aufwand zur Anpassung der Interaktionsmöglichkeiten bei der Integration bedeutet, was wiederum Anforderung A2 widerspricht. Daher ermöglicht auch dieser Ansatz zum Datenaustausch in einer Co-Simulation nur sehr beschränkt „Plug-and-Simulate“.

Da keines der existierenden Konzepte zum Datenaustausch uneingeschränkt „Plug-and-Simulate“ ermöglicht, wird für das oben vorgestellte Co-Simulationskonzept ein neues Datenaustauschkonzept entwickelt.

3.2.2 Datenaustausch in der Co-Simulationsumgebung

Wie schon in Kapitel 3.1.3 erwähnt, ist die Hauptaufgabe der Co-Simulationsumgebung die Interaktionen, die durch die Komponentensimulationen über die Komponentenvertreter versendet werden, an die Interaktionssimulation über deren Interaktionsvertreter weiterzuleiten. Hierzu muss die Interaktionssimulation eindeutig identifizierbar sein. Da diese allerdings durch ihren Interaktionsvertreter gekapselt ist, reicht es aus, wenn dieser eindeutig identifizierbar ist. Anschließend muss die Co-Simulationsumgebung die von der Interaktionssimulation verarbeiteten Interaktionen an die Zielkomponentensimulation bzw. deren Komponentenvertreter weiterleiten (siehe Abbildung 8: interaktionsorientierte Weiterleitung). Weil es sich hierbei meist um eine andere Komponentensimulation handelt, als die von der die Interaktion ausging, muss diese Zielkomponentensimulation ebenfalls eindeutig identifizierbar sein, damit die Interaktion den richtigen Adressaten finden kann. Hier reicht es aufgrund der Kapselung durch den Komponentenvertreter ebenfalls aus, dass nur dieser eindeutig identifizierbar ist. Somit besitzt die Co-Simulationsumgebung ein Adressregister, welches alle eindeutigen IDs der einzelnen Simulationsvertreter kennt und auf das bei einem Interaktionsaustausch zugegriffen werden kann (siehe Abbildung 8: Adressregister). Die interaktionsorientierte Weiterleitung wird in den Kapiteln 3.2 und 3.3 detailliert beschrieben.

Tritt eine neue Teilsimulation in die Co-Simulation ein, wird ein neuer Simulationsvertreter instanziiert. Dieser muss sich dann mit seiner ID am Adressregister anmelden.

Für die durch den Taktgeber koordinierte Synchronisation kann ebenfalls auf das Adressregister zugegriffen werden. Allerdings werden diese Nachrichten parallel zu den Interaktionsnachrichten

behandelt um eine klare Trennung zu erhalten (siehe Abbildung 8: synchronisationsorientierte Weiterleitung). Somit spaltet sich die Nachrichtenweiterleitung aus Kapitel 3.1.3 in die Interaktionsweiterleitung und die synchronisationsorientierte Weiterleitung auf. Die Synchronisation wird in Kapitel 3.4 detailliert beschrieben.

Um die oben beschriebene eindeutige Identifizierung zu ermöglichen muss jeder Simulationsvertreter eine eindeutige ID besitzen. (siehe Abbildung 8: ID)

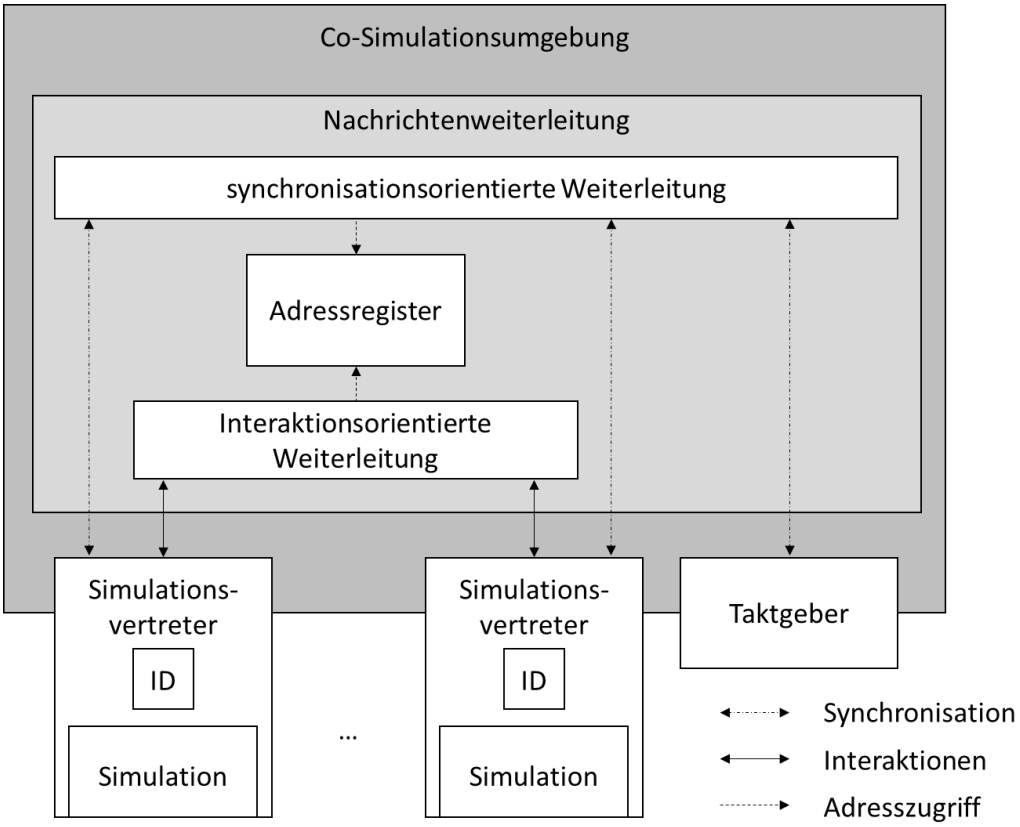


Abbildung 8: Datenaustausch in der Co-Simulationsumgebung

3.2.3 Datenaustauschschnittstellen

Um einen Datenaustausch zwischen den Teilsimulationen und ihren Simulationsvertretern zu ermöglichen wird eine Schnittstelle zwischen diesen benötigt (siehe Abbildung 9). Aufgrund der in Kapitel 3.1.1 getroffenen Entscheidung, einen Open-Source-orientierten Ansatz zu verfolgen muss diese Schnittstelle zusätzlich von außen an die Simulationstools angebracht werden.

Da es, wie in Kapitel 3.2.2 beschrieben, in der Co-Simulationsumgebung beim Datenaustausch eine Unterteilung in interaktionsorientierte Weiterleitung und synchronisationsorientierte Weiterleitung gibt, werden diese beiden Arten des Datenaustauschs ebenfalls in der Datenaustauschschnittstelle getrennt behandelt. Somit gibt es zwei voneinander getrennte Schnittstellen zwischen den Simulationstools und ihren Simulationsvertretern: die interaktionsorientierte Schnittstelle und die Synchronisationsschnittstelle (Abbildung 9).

Die Simulationsvertreter sollen eine Kapselung der einzelnen Teilsimulationen gegenüber der restlichen Co-Simulation bewirken. Daher ist es notwendig, dass alle Teilsimulationen auf die gleiche Art und Weise an die Co-Simulationsumgebung angebunden werden. Dadurch unterscheiden sich die einzelnen Simulationsvertreter bezüglich der Kommunikation über die Co-Simulationsumgebung nicht und der Teil der Datenaustauschschnittstelle der die Verbindung zu den Simulationsvertretern herstellt ist unabhängig vom darunterliegenden Simulationstool. Daher kann, um sowohl eine hohe Wiederverwendbarkeit als auch einen geringeren Aufwand beim Einbinden von neuen Simulationstools zu erreichen, die Datenaustauschschnittstelle in einen generischen und einen simulationstoolspezifischen Teil unterteilt werden (siehe Abbildung 9, hier am Beispiel eines Komponentenvertreters).

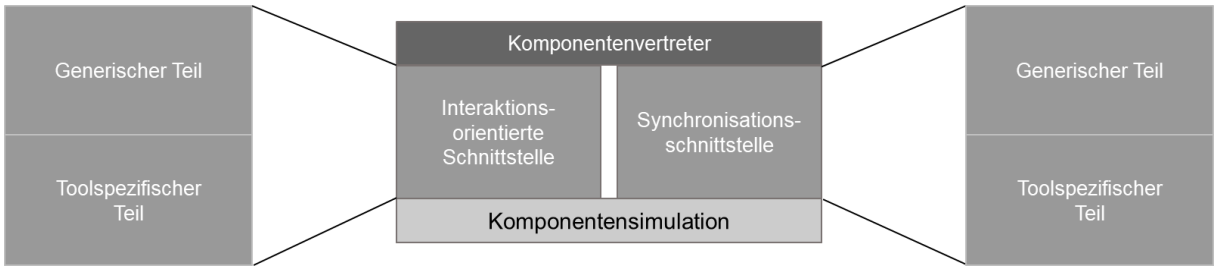


Abbildung 9: Konzept der Datenaustauschschnittstelle

Der generische Teil, der die Verbindung zu den Simulationsvertretern bildet, ist somit für alle Simulationstools gleich und kann immer wieder verwendet werden.

Der simulationstoolspezifische Teil hingegen muss für jedes Simulationstool implementiert werden, da er auf die vom Simulationstool bereitgestellten Schnittstellen zugreift. Hier ist allerdings das Ziel, dass dies nur für jedes Simulationstool und nicht für jedes Modell in einem Simulationstool geschehen muss. Diese Implementierung hängt von dem zu integrierenden Simulationstool ab, es ist aber auch hier vorgesehen, den simulationstoolspezifischen Teil modular aufzubauen, sodass bestimmte Teile für verschiedene Simulationstools immer wieder verwendet werden können, wenn sie ähnliche Schnittstellen oder Schnittstellen, die einem ähnlichen Prinzip folgen, haben. Beispiele für solche modularen Bausteine werden detaillierter in der Realisierung in Kapitel 4.2 vorgestellt.

Auch bei den simulationstoolspezifischen Schnittstellen ist eine Trennung in interaktionsorientierte und Synchronisationsschnittstelle vorgesehen. Bietet das gewählte Simulationstool allerdings nicht diese Möglichkeit, da Datenaustausch und Synchronisation über die gleiche Schnittstelle erfolgen, wird nur eine simulationstoolspezifische Schnittstelle umgesetzt und die Trennung in interaktionsorientierte und Synchronisationsschnittstelle im generischen Teil der Schnittstelle durchgeführt, siehe Abbildung 10. Hierfür wird ebenfalls eine detailliertere Beschreibung und ein Beispiel in der Realisierung in Kapitel 4.2.2 gegeben.

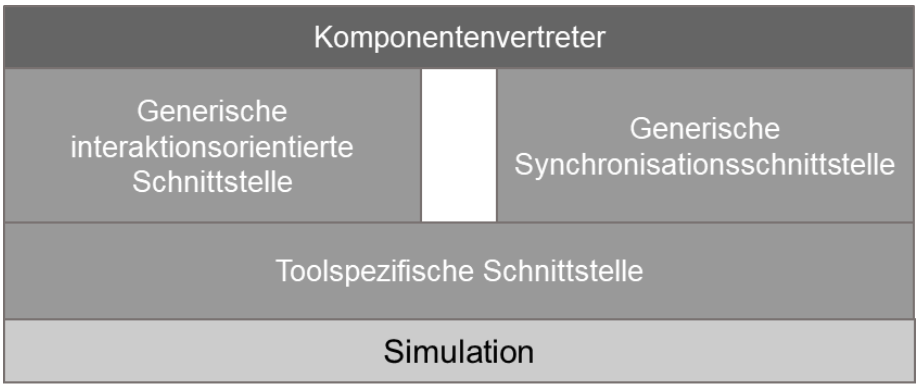


Abbildung 10: Datenaustauschschnittstelle mit einer simulationstoolspezifischen Schnittstelle

3.2.4 Anforderungen an die Teilsimulationen und ihre Simulationstools

Durch das gewählte Datenaustauschkonzept und die in Kapitel 3.1.1 getroffene Entscheidung, den Open-Source-orientierten Ansatz zu wählen, ergeben sich Anforderungen an die Schnittstellen, die die zu verwendenden Simulationstools bereitstellen müssen.

Zum einen muss eine Schnittstelle existieren, über die ein Datenaustausch stattfinden kann, idealerweise ohne die Simulation während des Datenaustauschs pausieren zu müssen. Sollte dies nicht der Fall sein, so kann der Datenaustausch nur in Synchronisationspausen stattfinden, hierfür wird in der Realisierung in Kapitel 5.2.2.3 ein Beispiel gegeben. Mit einem derartigen Simulationstool können nur solche Komponenten simuliert werden, die keine schnellen Antworten von anderen Komponenten benötigen und nicht allzu zeitkritisch sind.

Zum anderen muss das Simulationstool eine Schnittstelle bereitstellen, über die die Synchronisation stattfinden kann. Dies ist idealerweise eine von der für den Datenaustausch verwendeten getrennte Schnittstelle. Ist dies nicht möglich, kann auf die oben vorgestellte Lösung mit nur einer simulationstoolspezifischen Schnittstelle zurückgegriffen werden.

Diese beiden Bedingungen gelten jedoch immer für Simulationstools, die in einer Co-Simulation verwendet werden sollen, da ohne Datenaustausch und Synchronisation und den dazugehörenden Schnittstellen keine Co-Simulation möglich ist.

3.3 Interaktionssimulationen

Bisher wurden nur Aspekte des Datenaustauschs in der Co-Simulation betrachtet, welche grundsätzlich einen Datenaustausch ermöglichen, der Inhalt des Datenaustauschs wurde noch nicht betrachtet. Wie in Kapitel 3.1.2 beschrieben müssen die Interaktionen zwischen den einzelnen Komponentensimulationen ebenfalls modelliert und simuliert werden, wodurch der Inhalt der Interaktionen ebenfalls relevant ist. Daher wird zunächst untersucht, welche Interaktionen es in einem IoT-System geben kann. Anschließend wird ein Konzept vorgestellt,

mit dem diese Interaktionen modelliert und simuliert werden können, sodass eine „Plug-and-Simulate“-fähige Co-Simulation ermöglicht wird. Abschließend werden Anforderungen aufgezeigt, die sich durch dieses Konzept an die Teilsimulationen und deren Simulationstools ergeben.

3.3.1 Interaktionsarten im IoT

Die Neuheit von IoT-Systemen ist die Vernetzung von Dingen, also die Befähigung von Dingen untereinander zu kommunizieren. Diese Kommunikation zwischen den einzelnen Dingen, im folgenden Komponenten genannt, stellt eine Interaktion zwischen den Komponenten dar, die einen Einfluss auf die einzelnen Komponenten und somit auch auf das Gesamtsystem hat. Diese Kommunikation kann beispielsweise über WLAN oder Bluetooth ablaufen, siehe Abbildung 11.

Bei der Simulation von IoT-Systemen muss allerdings neben der Kommunikation zwischen den einzelnen Komponenten auch die prozessorientierte Interaktion zwischen den Komponenten betrachtet werden. Der Unterschied zwischen einer Interaktion per Kommunikation und einer prozessorientierten Interaktion soll an folgendem Beispiel verdeutlicht werden: Fordert ein Werkstück durch eine Kommunikationsnachricht eine Bearbeitungsstation auf, es zu bearbeiten, findet eine Interaktion zwischen Werkstück und Bearbeitungsstation per Kommunikation statt, welche den inneren Zustand beider beeinflusst. Die anschließende Bearbeitung des Werkstücks, beispielsweise Bohren, hingegen ist eine prozessorientierte Interaktion zwischen dem Bohrer und dem zu bearbeitenden Werkstück und beeinflusst ebenfalls den inneren Zustand von einer oder mehreren beteiligten Komponenten, siehe Abbildung 11. Die prozessorientierten Interaktionen bilden folglich die physikalischen Prozesse in einem IoT-System ab.

Prozessorientierte Interaktionen lassen sich in zwei Kategorien einteilen, eine direkte Interaktion zwischen zwei Komponenten und eine indirekte Interaktion einer Komponente über die Umwelt mit einer anderen Komponente. Ein Beispiel für die direkte Interaktion wäre eine Bearbeitung einer Komponente durch eine andere Komponente oder der Transport einer Komponente durch eine andere Komponente. Eine indirekte Interaktion über die Umwelt hingegen ist beispielsweise das Erhitzen der Umgebung durch eine Komponente und das anschließende Messen der Umgebungstemperatur durch eine andere Komponente (Abbildung 11).

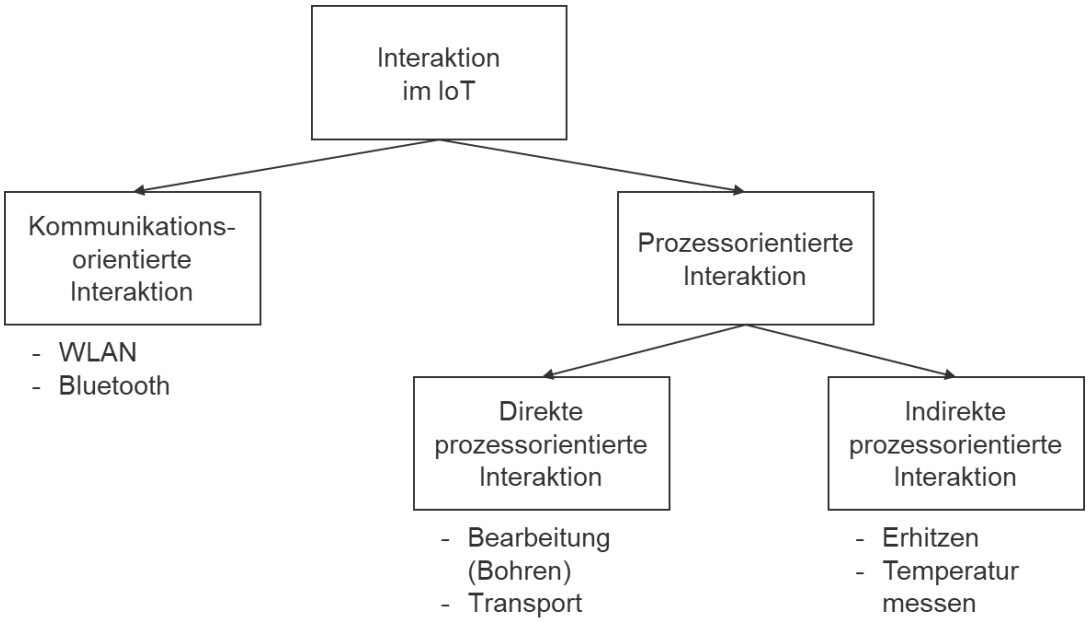


Abbildung 11: Interaktionsarten im IoT

Da, wie in Kapitel 3.1.2 beschrieben, die Interaktionen bzw. die Verarbeitung und Weiterleitung der Interaktionen in einer eigenen Simulation modelliert und simuliert werden, muss diese Simulation sowohl die kommunikationsorientierten als auch die prozessorientierten Interaktionen simulieren. Die meisten Simulationstools sind entweder auf Kommunikationsnetze, welche für die kommunikationsorientierten Interaktionen genutzt werden können, oder auf physikalische Prozesse, welche für die prozessorientierten Interaktionen genutzt werden können, spezialisiert, weshalb es nicht sinnvoll ist, nur ein einziges Simulationstool für die Interaktionssimulation zu verwenden. Daher wird die Interaktionssimulation in eine Kommunikationssimulation und eine Prozesssimulation aufgeteilt. Diese müssen jeweils von einem eigenen Interaktionsvertreter vertreten werden, die sich leicht voneinander unterscheiden. Es wird eine Unterteilung in Kommunikationsvertreter und Prozessvertreter vorgenommen, welche näher in Kapitel 3.3.2 und Kapitel 3.3.3 erläutert werden. Somit wird in dem in Kapitel 3.1.2 vorgestellten Konzept die Interaktionssimulation durch eine Kommunikationssimulation und eine Prozesssimulation ersetzt, Abbildung 12.

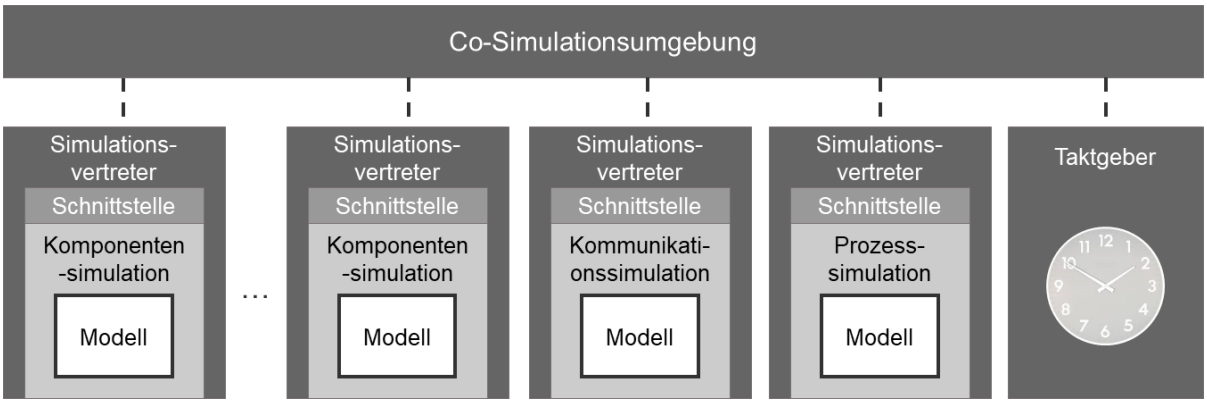


Abbildung 12: Co-Simulation mit Kommunikationssimulation und Prozesssimulation

Ist im Weiteren die Rede von Interaktionsvertreter, so gilt die Aussage sowohl für Kommunikations- als auch für Prozessvertreter.

Dies bedeutet für die Co-Simulationsumgebung, dass sich die interaktionsorientierte Weiterleitung in eine kommunikationsorientierte und eine prozessorientierte Weiterleitung aufteilt, dargestellt in Abbildung 13.

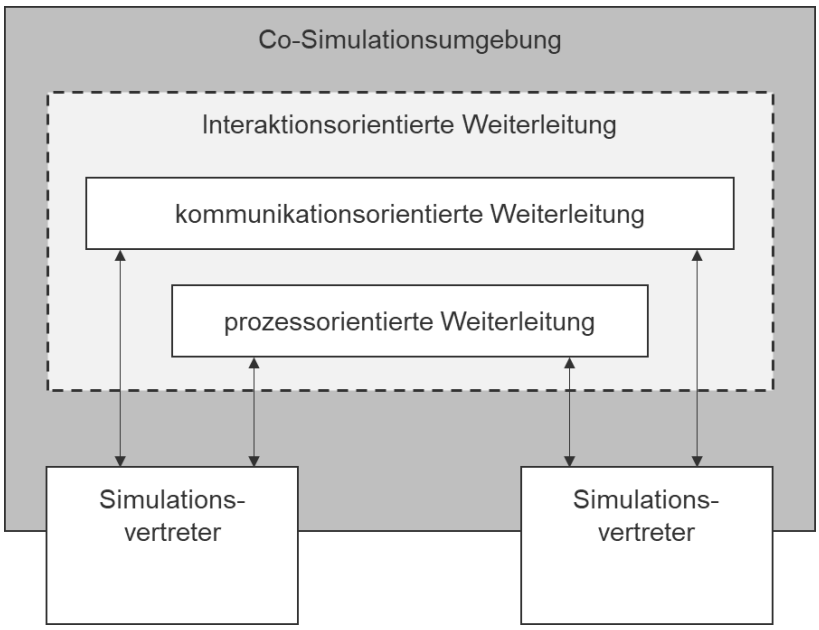


Abbildung 13: Erweiterte Co-Simulationsumgebung

Ebenso wird die interaktionsorientierte Datenaustauschschnittstelle in eine kommunikationsorientierte und eine prozessorientierte Datenaustauschschnittstelle aufgeteilt, dargestellt in Abbildung 14 am Beispiel eines Komponentenvertreters.

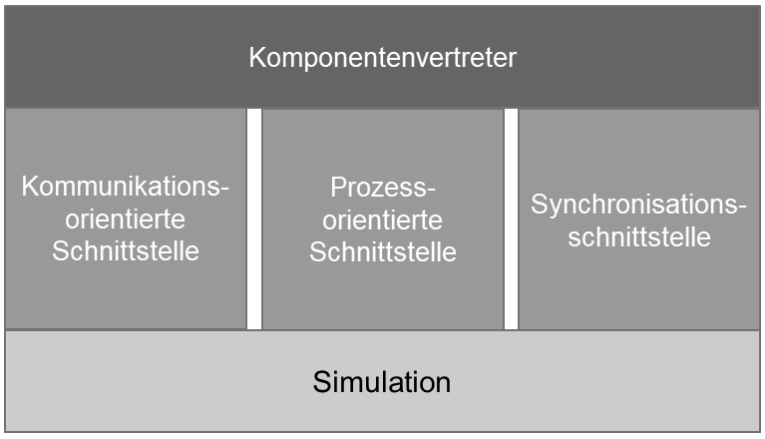


Abbildung 14: Erweiterte Datenaustauschschnittstelle

In den beiden folgenden Unterkapiteln 3.3.2 und 3.3.3 werden Konzepte vorgestellt, wie diese Interaktionsarten in einer „Plug-and-Simulate“-fähigen Co-Simulation umgesetzt werden können.

3.3.2 Kommunikationsorientierte Interaktionssimulation

Um eine kommunikationsorientierte Interaktionssimulation in einer dynamischen Co-Simulation eines IoT-Systems zu ermöglichen, werden folgende Bestandteile benötigt, bzw. sind folgende Anpassungen an den bereits beschriebenen Bestandteilen, zusammengefasst in Tabelle 5.

Tabelle 5: Anpassungen zur kommunikationsorientierten Interaktionssimulation

Komponente	**Ziel**	**Aufgabe**
Kommunikations-simulation	Spezifikation der Kommunikations-interaktionen	Spezifikation der Austauschart der Kommunikationsinteraktions-nachrichten Spezifikation der Kommunikationsinteraktions-nachrichten
Kommunikations-simulation	Modellierung der Kommunikations-simulation	Modellierung und Simulation der verschiedenen Kommunikationskanäle in einem Simulationstool
Kommunikations-simulation	Zuweisung einer Nachricht zu einem Knoten in der Kommunikations-simulation	Spezifikation der Weiterleitung einer Nachricht vom Kommunikations-vertreter zur richtigen Position in der Kommunikationssimulation
Simulationsvertreter	Erstellung eines Kommunikations-vertreters	Erstellung Konzept des Kommunikationsvertreters Anpassung der Komponentenvertreter
Schnittstellen zwischen Simulationsvertreter und Simulation	Erstellung einer kommunikations-orientierten Interaktionsschnittstelle	Erstellung Konzept der kommunikationsorientierten Schnittstellen sowohl der Komponentenvertreter als auch der Kommunikationsvertreter
Kommunikations-simulation	Verwendung mehrerer Kommunikations-simulation	Erstellung Konzept zur Verwendung unterschiedlicher Kommunikations-simulationen in unterschiedlichen Simulationstools

Bevor diese einzelnen Bestandteile im Folgenden beschrieben werden, wird zunächst der typische Ablauf eines Austauschs einer kommunikationsorientierten Interaktion erläutert (siehe Abbildung 15): Zuerst wird die zu versendende Nachricht in einer Komponentensimulation erzeugt und über die kommunikationsorientierte Schnittstelle an den Komponentenvertreter weitergegeben. Dieser versendet dann die Nachricht an den Kommunikationsvertreter, welcher sie an die Kommunikationssimulation weiterleitet. Diese verarbeitet die Nachricht und simuliert die Übertragung an den Empfänger, anschließend gibt sie die Nachricht zurück an den Kommunikationsvertreter, welcher sie dann an den empfangenden Komponentenvertreter sendet. Dieser gibt sie abschließend an seine Komponentensimulation weiter.

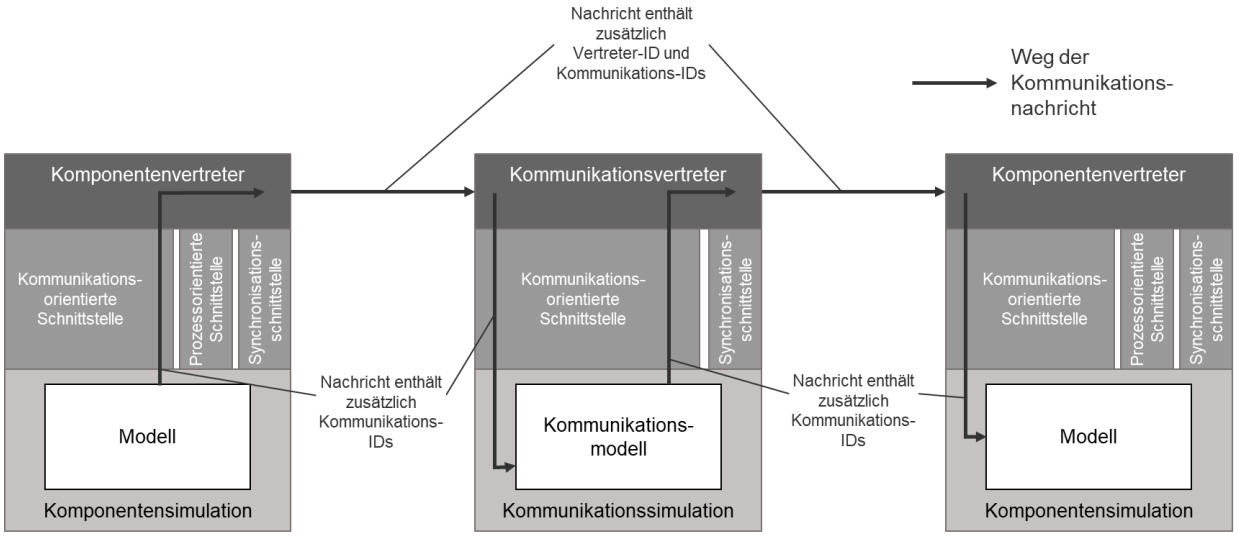


Abbildung 15: Weg einer Kommunikationsnachricht

3.3.2.1 Spezifikation der Kommunikationsinteraktionen

Damit eine von einer Komponentensimulation versendete Kommunikationsnachricht von der Kommunikationssimulation verarbeitet und von der empfangenden Komponentensimulation verstanden und interpretiert werden kann, ist es notwendig, dass die Form der Nachrichten spezifiziert ist. Die Nachricht, die von der Komponentensimulation zu ihrem Komponentenvertreter übermittelt wird, ist eine simulierte Nachricht, wie sie in dem realen System vorkommen kann. Das heißt sie folgt exakt der Spezifikation der verwendeten Kommunikationstechnologie, wie beispielsweise der Spezifikation von Wi-Fi [175], in der genau beschrieben ist, wie die Nachricht aufgebaut ist. Sie enthält neben dem Nachrichteninhalt auch die Identifikationsmechanismen zur Identifikation von Sender und Empfänger (siehe Abbildung 16), im Weiteren Kommunikations-IDs genannt (Abbildung 15). Liegt bei der Kommunikation kein Single-Cast vor, sondern ein Multi-Cast, so enthält die Nachricht mehrere Empfänger-IDs (Kommunikations-IDs) und es müssen mehrere Vertreter-IDs angefügt werden. Falls es keine Kommunikations-ID gibt, wie beispielsweise bei Broadcasts oder bei „Publish-Subscribe-Schemas“, so werden entweder die Nachrichten an alle Simulationsvertreter oder als Antwort an den Sender gesendet. Für diesen Falls muss sich der Kommunikationsvertreter merken, von

welchem Simulationsvertreter die Nachricht kam. Somit können unterschiedlichste Kommunikationsverfahren in der Co-Simulationsumgebung abgebildet werden. Ist die Nachricht im Komponentenvertreter angekommen, so fügt dieser der Nachricht seine Vertreter-ID hinzu, damit der Kommunikationsvertreter identifizieren kann, von welcher Komponentensimulation die Nachricht stammt, um sie richtig an die Kommunikationssimulation zu übergeben. Dies ist notwendig, da die Simulationsvertreter durch die Kapselung keine Informationen über die Simulationsinhalte und somit auch keine Informationen über die Nachrichteninhalte oder Nachrichtensender und -empfänger haben. Wird die Nachricht nach der Simulation der Nachrichtenübermittlung von der Kommunikationssimulation an den Kommunikationsvertreter zurückgegeben, so fügt dieser die Vertreter-ID des Empfängervertreters der Nachricht hinzu. Dies geschieht durch ein Mapping von Kommunikations-IDs zu Vertreter-IDs, näher beschrieben in 3.3.2.5. Somit besitzt die Nachricht für den Austausch unter Simulationsvertretern noch zusätzlich eine Vertreter-ID, siehe Abbildung 16. Im Fall, dass die Nachricht zu einem Kommunikationsvertreter geht, ist es die Vertreter-ID des sendenden Komponentenvertreters, im Fall, dass sie von einem Kommunikationsvertreter zu einem Komponentenvertreter geht, die des empfangenden Komponentenvertreters. Von den Nachrichten, die von den Simulationsvertretern an ihre Simulationen übergeben werden, werden die Vertreter-IDs entfernt, so dass nur die Nachrichten, die der Spezifikation der Nachrichtentechnologie entsprechen an der Simulation ankommen. Der Nachrichteninhalt selbst wird dann in den Simulationen extrahiert.

Abbildung 16: Kommunikationsnachrichtenspezifikation

3.3.2.2 Modellierung der Kommunikationssimulation

Die Kommunikationssimulation hat die Aufgabe die Nachrichtenübertragung zu simulieren, wobei Effekte wie Nachrichtenverluste, Dämpfungen in den Übertragungskanälen, Verzerrungen und ähnliches simuliert werden können. Wie detailliert diese Simulation ist, hängt vom Anwendungsfall und der simulierten Kommunikationstechnologie ab.

In der Modellierung muss zu jeder Komponentensimulation in der Co-Simulation ein Kommunikationsknoten in der Kommunikationssimulation existieren, so dass jeder Komponentensimulation ein Knoten in der Kommunikationssimulation zugeordnet werden kann. Dies ist wichtig für das Mapping zwischen Kommunikations-IDs und Vertreter-IDs.

Der grundsätzliche Ablauf ist für jede Nachricht gleich: vom Kommunikationsvertreter kommt eine Nachricht in die Kommunikationssimulation und wird dem entsprechenden Knoten

zugeordnet, daraufhin erfolgt die Simulation der Nachrichtenübertragung, die damit endet, dass die Nachricht am Empfängerknoten ankommt. Von diesem muss sie dann zurück an den Kommunikationsvertreter übermittelt werden. Hierbei muss es möglich sein, dem Kommunikationsvertreter mitzuteilen, welcher Knoten genau dies ist, um ein Mapping durchführen zu können. Somit geht die Nachricht in der Simulation von Senderknoten zu Empfängerknoten, wie sie in der Realität von Senderkomponente zu Empfängerkomponente gehen würde.

Die Zuordnung der Kommunikations-IDs zu den einzelnen Komponentensimulationen erfolgt nach der Spezifikation der Kommunikationstechnologien.

Weiterhin muss in dem Simulationstool und in der Modellierung der Kommunikation vorgesehen sein, dass neue Knoten zur Simulationszeit hinzugefügt werden können, um den Eintritt von Komponenten zur Laufzeit simulieren zu können. Das Anlegen eines neuen Knoten wird durch eine Instanziierungsnachricht getriggert, welche von der neuen Komponentensimulation versendet wird. Hierbei wird der Verbindungsaufbaumechanismus der Kommunikationstechnologie genutzt, das heißt, der Verbindungsaufbau mit einer neuen Komponente funktioniert genauso in der Simulation, wie er auch in der Realität funktioniert.

Für den Fall, dass unterschiedliche Kommunikationstechnologien verwendet werden, empfängt jede Kommunikationssimulation die Instanziierungsnachricht, für den Fall, dass diese ihrer Kommunikationstechnologie entspricht, quittiert sie über an den Kommunikationsvertreter, dass erfolgreich ein neuer Knoten angelegt wurde, falls nicht, quittiert sie einen Fehlschlag.

Die Kommunikationssimulation ist somit ebenfalls komplett durch ihren Kommunikationsvertreter gekapselt.

3.3.2.3 Kommunikationsorientierte Schnittstelle

Die kommunikationsorientierten Schnittstellen der Kommunikationssimulation und einer Komponentensimulation unterscheiden sich leicht.

Kommunikationsorientierte Schnittstelle der Kommunikationssimulation

Die kommunikationsorientierte Schnittstelle der Kommunikationssimulation muss wie in 3.2.3 beschrieben, eine Verbindung zu einem Simulationstool über einen spezifischen Teil und zum Kommunikationsvertreter über einen generischen Teil aufnehmen. Dies ist gleich wie bei einer Komponentensimulation, bei der Kommunikationssimulation muss aber zusätzlich eine Instanziierungsnachricht an die Kommunikationssimulation weitergeleitet werden, welche das Anlegen eines neuen Knotens in der Kommunikationssimulation veranlasst. Zusätzlich muss die Quittierung an den Kommunikationsvertreter zurückgegeben werden können (Abbildung 17, Instanziierung und Quittierung).

Zudem muss die Möglichkeit bestehen, eine Nachricht, die vom Kommunikationsvertreter kommt gezielt einem Knoten in der Kommunikationssimulation zuzuweisen und zu identifizieren, von welchem Knoten die Nachricht von der Kommunikationssimulation kam (Abbildung 17, Eingangs- und Ausgangsknoten). Wie dies im Detail implementiert wird, hängt von dem verwendeten Simulationstool ab.

Sollte eine Kommunikationssimulation mehrere Kommunikationstechnologien simulieren, so ist eine Trennung der Kommunikationstechnologien in der Schnittstelle zwingend, da sonst ein Mapping nicht möglich ist, siehe Abbildung 17 (Schnittstelle Komm-Technologie 1 und Schnittstelle Komm-Technologie 2). Werden mehrere Kommunikationstechnologien simuliert, so muss beim Einbinden der Kommunikationssimulation für jede Schnittstelle ein Kommunikationsdienst angelegt und der entsprechenden Schnittstelle zugeordnet werden. Dies kann über ein Auslesen der verbundenen Schnittstellen erfolgen.

Bei einer Kommunikationssimulation kann auf die prozessorientierte Schnittstelle verzichtet werden, da keine prozessorientierten Nachrichten empfangen und versendet werden.

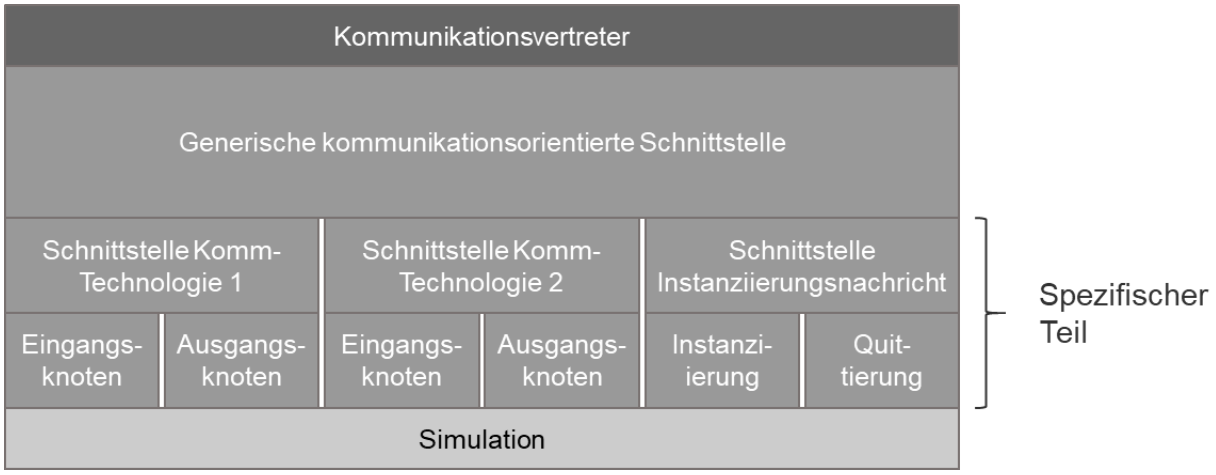


Abbildung 17: Kommunikationsorientierte Schnittstelle einer Kommunikationssimulation

Kommunikationsorientierte Schnittstelle der Komponentensimulation

Die kommunikationsorientierte Schnittstelle der Komponentensimulation muss wie in 3.2.3 beschrieben, eine Verbindung zu einem Simulationstool über einen spezifischen Teil und zum Komponentenvertreter über einen generischen Teil aufnehmen. Die Schnittstelle muss noch für den Fall erweitert werden, dass eine Komponente mehrere Kommunikationstechnologien verwendet: es muss möglich sein zu identifizieren, welcher Kommunikationstechnologie eine versendete Nachricht angehört, damit sie an die richtige Kommunikationssimulation versendet werden kann. Die Implementierung hängt ebenfalls vom gewählten Simulationstool ab, wobei es idealerweise möglich ist den spezifischen Teil aufzuteilen, sodass er auf unterschiedliche Schnittstellen des Simulationstools zugreift und somit zu einer sauberen Schnittstellentrennung führt.

Zusätzlich muss diese kommunikationsorientierte Schnittstelle ebenfalls eine Instanziierungsnachricht ermöglichen. Dies geschieht dadurch, dass beim Einbinden einer neuen Teilsimulation und somit beim Verbinden mit der kommunikationsorientierten Schnittstelle eine Instanziierungsnachricht gemäß der Spezifikation der Kommunikationstechnologie durch die kommunikationsorientierte Schnittstelle getriggert und der Verbindungsaufbau einer Komponente mit dem Kommunikationsnetzwerk simuliert wird, wie er in der Realität ablaufen würde. Diese Instanziierungsnachricht wird an die Kommunikationssimulation gesendet, welche diese für das Erstellen eines neuen Knotens benötigt.

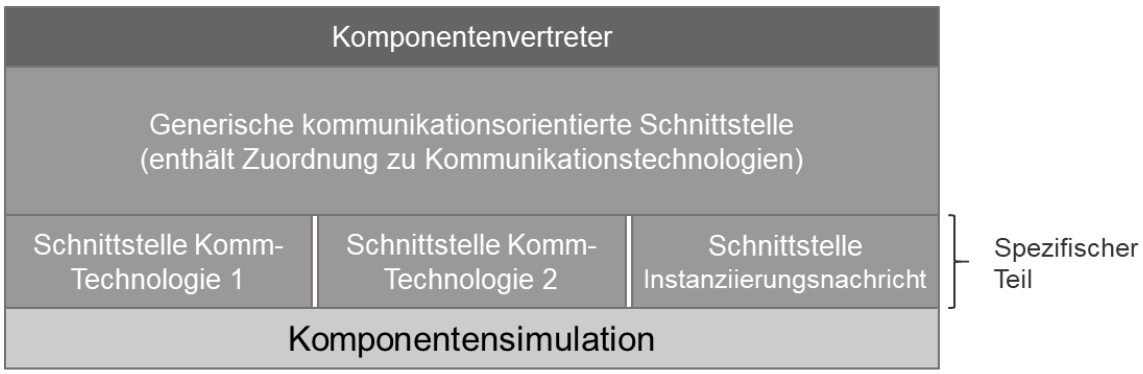


Abbildung 18: Kommunikationsorientierte Schnittstelle einer Komponentensimulation

3.3.2.4 Kommunikations- und Komponentenvertreter

Wie schon in 3.1.2 erwähnt, gibt es einen Unterschied zwischen Komponentenvertretern und Interaktionsvertretern (Kommunikations- und Prozessvertreter).

Erweiterung der Komponentenvertreter

Da möglicherweise eine Komponente unterschiedliche Kommunikationstechnologien einsetzt, bedarf es nicht nur in der Schnittstelle einer Unterscheidung der Nachrichten nach Kommunikationstechnologie, sondern auch im Komponentenvertreter. Dies geschieht dadurch, dass beim Verbinden mit dem Simulationstool die Instanziierungsnachrichten für jede Kommunikationstechnologie getriggert werden. Diese werden an alle in der Co-Simulation vorhandenen Kommunikationsvertreter versendet. Der Kommunikationsvertreter der mit diesen Instanziierungsnachrichten erfolgreich einen Knoten in seiner Simulation anlegen kann, quittiert dies dem Komponentenvertreter. Somit kann dieser eine Liste der möglichen Kommunikationstechnologien anlegen, in der die Kommunikationsvertreter den unterschiedlichen Nachrichten aus der Komponentensimulation zuordnet sind (siehe Abbildung 19).

Um ein Quittieren und eine Unterscheidung zwischen Instanziierung- und Inhaltsnachricht zu erleichtern wird der Nachrichtenaustausch von einem Komponentenvertreter zu einem Kommunikationsvertreter in Diensten aufgebaut. Somit bietet jeder Kommunikationsvertreter einen oder mehrere Kommunikationsdienste an und kann bei einem Dienstaufruf eines

Komponentenvertreters eine Quittierung als Antwort auf diesen Dienstaufruf geben. Zudem kann jeweils eine Instanziierungsnachricht somit an alle Instanziierungsdienste aller Kommunikationsdienste aller Kommunikationsvertreter versendet werden, wodurch diese gezielter zugestellt werden können. Ein Komponentenvertreter selbst bietet keine Dienste an, um die Wiederverwendbarkeit der Komponentenvertreter zu erhöhen und damit beim Eintritt eines Komponentenvertreters keine spezifischen Änderungen vorgenommen werden müssen.

In dem Adressregister der Co-Simulationsumgebung müssen daher nicht nur die Vertreter-IDs vermerkt sein, sondern auch welcher Art der Vertreter ist (Komponente, Kommunikation oder Prozess) und welche Dienste er anbietet (siehe Abbildung 19).

Kommunikationsvertreter:

Die Hauptaufgaben der Kommunikationsvertreter bestehen darin, den Komponentenvertretern die Kommunikationsdienste anzubieten und ein Mapping zwischen Vertreter-ID und Kommunikations-ID zu ermöglichen. Hierfür benötigt er eine Liste seiner Kommunikationsdienste, siehe Abbildung 19. Zudem muss er Nachrichten an Empfängerkomponentenvertreter weiterleiten, was über die Vertreter-ID erfolgt, die durch das im nächsten Abschnitt beschriebene Mapping ermittelt wird.

Tritt ein Kommunikationsvertreter neu in die Co-Simulation ein, so muss er dem Adressregister neben seiner ID auch die von ihm angebotenen Dienste mitteilen. Für jede von ihm vertretene Kommunikationstechnologie bietet er einen Kommunikationsdienst und einen Instanziierungsdienst an.

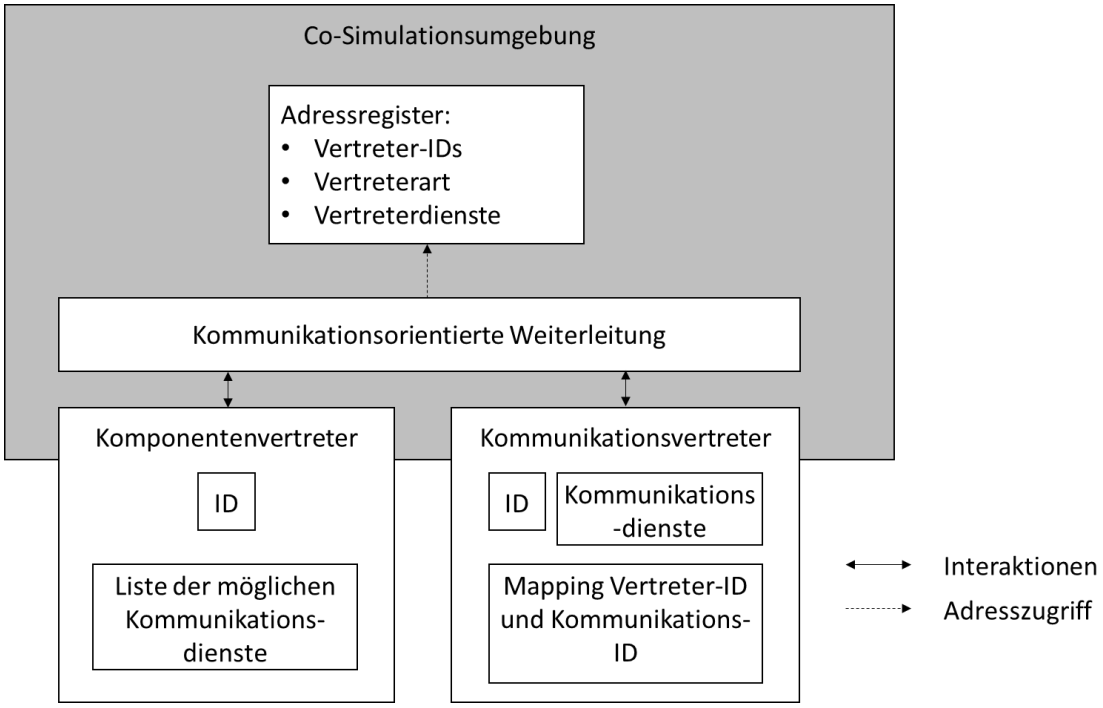


Abbildung 19: Co-Simulationsumgebung mit Kommunikationsvertreter

3.3.2.5 Mapping Vertreter-ID mit Kommunikations-ID

Ein Mapping zwischen Vertreter-ID und Kommunikations-ID muss in drei Fällen erfolgen: Instanziieren eines neuen Knoten, Empfangen einer Nachricht und Versenden einer Nachricht:

Instanziieren eines neuen Knoten

Wird eine Instanziierungsnachricht von einem Kommunikationsvertreter empfangen, wird dessen Vertreter-ID in einer Liste im Kommunikationsvertreter abgelegt. Für jede Kommunikationstechnologie existiert eine eigene Liste. Anschließend wird die Instanziierungsnachricht an die Simulation weitergegeben, in der ein Knoten mit einer ID angelegt wird, diese Knoten-ID kann identisch mit der Kommunikations-ID des Versenders in der Kommunikationstechnologie sein. Ist sie das nicht, muss in der Simulation eine Zuordnung von Knoten-ID zu Kommunikations-ID erfolgen. Ist das Anlegen des Knotens erfolgreich, erfolgt eine positive Quittung und eine Zuordnung zwischen dem Eintrag der Vertreter-ID und der Knoten-ID.

Empfangen einer Nachricht

Wird eine Nachricht über einen Kommunikationsdienst empfangen, wird in der angelegten Liste die zu der Vertreter-ID des Versenders gehörende Knoten-ID ermittelt und diesem Knoten in der Kommunikationssimulation der zugehörigen Kommunikationstechnologie über die kommunikationsorientierte Schnittstelle die Nachricht übermittelt. Diese Nachricht enthält nicht mehr die Vertreter-ID.

Versenden einer Nachricht

Beim Übermitteln einer Nachricht von der Kommunikationssimulation an den Kommunikationsvertreter, wird gleichzeitig die Knoten-ID des Empfängerknotens übermittelt (die Nachricht geht in der Kommunikationssimulation von Senderknoten zu Empfängerknoten). Somit kann der Kommunikationsvertreter in der Liste der zugehörigen Kommunikationstechnologie die Vertreter-ID des Empfängerknotens ermitteln und an diesen die Nachricht übermitteln.

3.3.2.6 Verwendung mehrerer Kommunikationssimulationen

Das Konzept erlaubt die Verwendung mehrerer Kommunikationssimulationen. Dies ist dann sinnvoll, wenn unterschiedliche Kommunikationstechnologien in einem IoT-System eingesetzt werden, die nicht alle in einer Simulation simulierbar sind oder wenn das gewählte Simulationstool nicht mehrere getrennte Schnittstellen zur Verfügung stellt, und somit kein Mapping zwischen Vertreter-ID und Kommunikations-ID möglich ist.

Prinzipiell ist es empfehlenswert für unterschiedliche Kommunikationstechnologien unterschiedliche Kommunikationssimulationen zu verwenden, da dies das Anbinden an den

Kommunikationsvertreter und das Mapping deutlich vereinfacht. Eine Zuordnung, an wen welche Kommunikationsnachricht gehen muss, erfolgt durch die im Adressregister verzeichneten Dienste.

3.3.3 Prozessorientierte Interaktionssimulation

Um eine prozessorientierte Interaktionssimulation in einer dynamischen Co-Simulation eines IoT-Systems zu ermöglichen, werden folgende Bestandteile benötigt, bzw. sind folgende Anpassungen an den bereits beschriebenen Bestandteilen nötig, welche in Tabelle 6 zusammengefasst sind.

Tabelle 6: Anpassungen zur Prozessorientierten Interaktionssimulation

Komponente	**Ziel**	**Aufgabe**
Prozesssimulation	Spezifikation der Prozessinteraktionen	Spezifikation der Austauschart der Prozessinteraktionsnachrichten Spezifikation der Prozessinteraktionsnachrichten
Prozesssimulation	Modellierung der Prozesssimulation	Modellierung und Simulation der verschiedenen physikalischen Prozesse in einem Simulationstool
Prozesssimulation	Zuweisung einer Interaktion zu einem Objekt in der Prozesssimulation	Spezifikation der Weiterleitung einer Interaktion von einem Prozessvertreter zur richtigen Position in der Prozesssimulation
Simulationsvertreter	Erstellung eines Prozessvertreters	Erstellung Konzept des Prozessvertreters Anpassung der Komponentenvertreter
Schnittstellen zwischen Simulationsvertreter und Simulation	Erstellung einer prozessorientierten Interaktionsschnittstelle	Erstellung Konzept der prozessorientierten Schnittstellen sowohl der Komponentenvertreter als auch der Prozessvertreter
Prozesssimulation	Verwendung mehrerer Prozesssimulationen	Erstellung Konzept zur Verwendung unterschiedlicher Prozesssimulationen in unterschiedlichen Simulationstools

Bevor diese einzelnen Bestandteile im Folgenden beschrieben werden, wird zunächst der typische Ablauf eines Austauschs einer prozessorientierten Interaktion erläutert (siehe Abbildung 20): Zuerst wird die Interaktion in einer Komponentensimulation erzeugt und über die prozessorientierte Schnittstelle an den Komponentenvertreter weitergegeben. Dieser versendet dann die Interaktion, indem er den entsprechenden Dienst (Dienste sind in 3.3.3.1 beschrieben) des Prozessvertreters aufruft, welcher sie an die Prozesssimulation über die prozessorientierte Schnittstelle weiterleitet. Diese verarbeitet die Interaktion und gibt eine eventuelle Antwort auf die Interaktion über die prozessorientierte Schnittstelle an den Prozessvertreter zurück. Dieser leitet sie an den Vertreter des Empfängers dieser Antwort weiter, welcher sie abschließend an seine Komponentensimulation weitergibt.

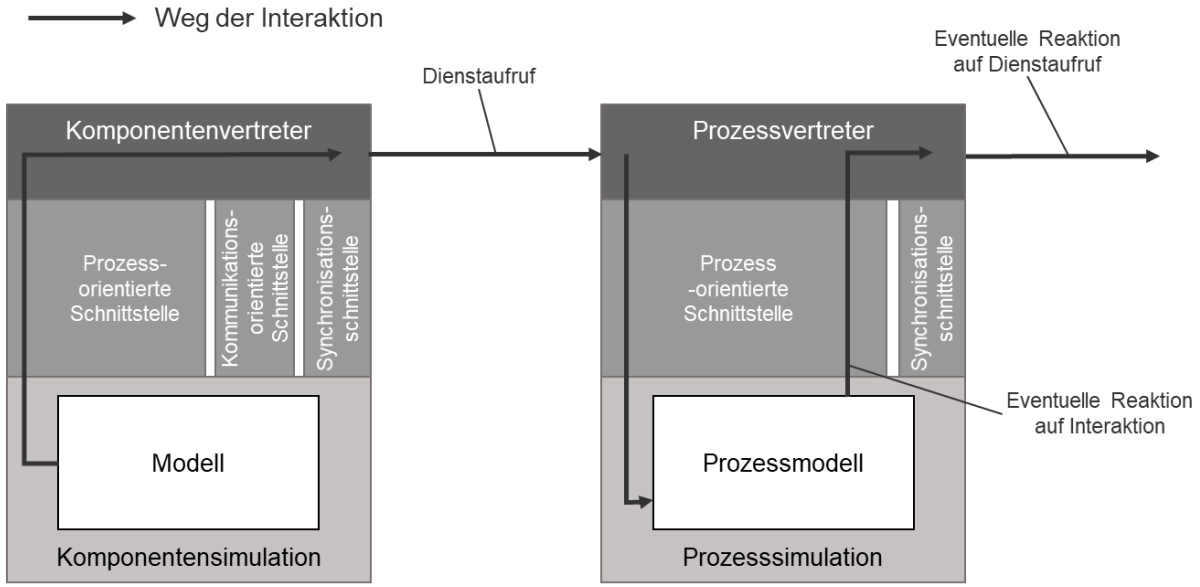


Abbildung 20: Prozessorientierter Interaktionsablauf

Es kommt auf die Interaktion an, ob es eine Antwort auf die Interaktion gibt und an wen diese geht:

- Keine Antwort gibt es beispielsweise bei der Interaktion erhitzen, hier wird nur der Prozesssimulation mitgeteilt, dass sich die Umgebung erhitzt, was von dieser verarbeitet wird.
- Eine Antwort an den Interaktionsinitiator, gibt es bei Interaktionen die messen, wie beispielsweise Temperatur messen.
- Eine Antwort an eine weitere Komponentensimulation gibt es beispielsweise bei der Interaktion Bewegen, bei der eine Komponente eine andere Komponente bewegt.

Um Antworten möglichst einfach verwalten zu können, wird bei der prozessorientierten Interaktion ein ähnliches Dienstesystem wie bei der kommunikationsorientierten Interaktion

eingesetzt, so dass Interaktionen, die von einem Komponentenvertreter an einen Prozessvertreter verschickt werden, über Dienste übermittelt werden.

3.3.3.1 Spezifikation der Prozessinteraktionen

Bei einer prozessorientierten Interaktion ist es nicht möglich, auf eine vorgefertigte Spezifikation wie bei einer kommunikationsorientierten Interaktion zurückzugreifen. Daher muss in der Co-Simulation eine eigene Spezifikation angefertigt werden. Dies kann eine Aufgabe der Prozesssimulation sein. Diese Spezifikation gibt sowohl den Abstraktionsgrad (es ist nicht möglich eine Prozessinteraktion bis ins letzte Detail zu simulieren, daher muss sie abstrahiert werden) als auch die benötigten Prozessparameter vor. Zudem wird spezifiziert, welche Prozesse über die Prozesssimulation simulierbar sind. Die Spezifikation erfolgt über eine Liste von Diensten, wobei jeder Dienst für einen Prozess in der Simulation verantwortlich ist und die benötigten Parameter definiert. In Tabelle 7 sind ein paar Beispiele für solche Dienste gegeben.

Tabelle 7: Beispiele für prozessorientierte Dienste

Dienst	Parameter	Interaktionsart	Beschreibung
messe_Temp	Koordinaten x, y, z	Indirekt	Gibt bei Aufruf die aktuelle Temperatur der Umgebung am angegebenen Ort (x,y,z) zurück.
erhitze	Leistung in KW	Indirekt	Erhitzt die Umgebung mit der übermittelten Leistung.
bewege	Richtung x, y, z und Geschwindigkeit v	Direkt	Bewegt die Komponente mit der angegebenen Geschwindigkeit in die angegebene Richtung.
hinzufügen	Position x, y, z Ausrichtung α und β Geometrie-Modell	-	Fügt eine neue Komponente an der angegebenen Position mit der angegebenen Ausrichtung der Prozesssimulation hinzu.

3.3.3.2 Modellierung der Prozesssimulation

Jede Komponente die einen physikalischen Einfluss auf andere Komponenten oder die Umgebung haben kann, muss in der Prozesssimulation abgebildet sein. Hierzu sollten, um direkte Interaktionen wie Kollisionen zwischen Komponenten simulieren zu können, die Abmessungen sowie die Bewegungsachsen der Komponenten modelliert werden. Diese Informationen müssen von den einzelnen Komponentenmodellen kommen, dies kann beispielsweise über standardisierte Modelltypen wie STEP [176] oder etwas Vergleichbares geschehen, die zusätzlich zu den Modellen mitgeliefert werden und von der Prozesssimulation gelesen werden können. Somit muss es in dem gewählten Simulationstool und in der Modellierung der Umgebung möglich sein,

Objekte zur Laufzeit hinzuzufügen, um einen Eintritt einer Komponente simulieren zu können. Dies muss beim Einbinden einer neuen Komponentensimulation geschehen, sodass in der Prozesssimulation ein Objekt mit den entsprechenden Eigenschaften erzeugt werden kann. Hier ist ein manueller Aufwand nötig, da angegeben werden muss an welcher Position und mit welcher Ausrichtung diese Komponente eingefügt wird. Dies widerspricht nicht Anforderung A1, da in dem realen System das physische Einfügen einer Komponente normalerweise ebenfalls einen manuellen Aufwand erfordert. Kommt die Komponente im realen System autonom in das System, so erfolgt dies über einen bestimmten Eingang in das System, was wiederum ebenfalls in der Prozesssimulation modelliert werden kann, wodurch in diesem Spezialfall kein manueller Aufwand entsteht.

Wird ein neues Objekt in der Prozesssimulation erzeugt, muss dieses eine eindeutige ID erhalten, damit ihm später während der Simulation Interaktionen zugeordnet werden können.

Ändert sich der Zustand eines Objekts entweder durch die Umgebung oder durch andere Objekte, muss dies erfasst werden. In der Realität wird dies über Sensoren erfasst, daher reicht es aus, dass nur solche Zustandsänderungen erfasst werden, welche von den in den Komponenten modellierten Sensoren erfasst werden. Diese erkannten Zustandsänderungen können als Reaktionen auf Dienste an die Komponentensimulationen zurückgesendet werden. In der Realität messen die meisten Sensoren entweder dauerhaft oder zyklisch, daher wird dieses Prinzip auch auf die Co-Simulation übertragen und die Komponentensimulationen bzw. deren simulierte Sensoren müssen regelmäßig bei der Prozesssimulation den aktuellen Zustand über einen Dienstaufruf erfragen.

Grundsätzlich ist jedoch auf eine klare Trennung bezüglich der Modellierung in den Komponentensimulationen und den Prozesssimulationen zu achten. Komponentensimulationen sollten keine Modellierung von physikalischen Prozessen enthalten, die andere Komponenten beeinflussen. Würden sie auch solche physikalischen Prozesse enthalten, müsste eine Übertragung der Daten dieser Prozesse an die Prozesssimulation stattfinden, was die Datenübertragung in der Co-Simulation bedeutend erschwert.

3.3.3.3 Zuweisung einer Interaktion zu einem Objekt in der Prozesssimulation

Es gibt zwei Fälle in denen eine Interaktion einem Objekt in der Prozesssimulation zugewiesen werden muss: beim Eintritt einer neuen Komponentensimulation und beim Aufruf eines Prozessdienstes.

Eintritt einer neuen Komponentensimulation

Tritt eine neue Komponentensimulation in die Co-Simulation ein, so ruft der Komponentenvertreter den Instanziierungsdienst des Prozessvertreters auf und übermittelt Position, Ausrichtung und Geometrie-Modell. Diese Parameter werden benötigt, um Kollisionen mit sich bewegenden Objekten simulieren zu können. Sollte das Geometriemodell nicht

vorhanden sein, so wird ein Platzhalter mit den ungefähren Abmaßen eingefügt. Daraufhin wird mit diesen Informationen ein neues Objekt mit einer Objekt-ID angelegt. Diese Objekt-ID wird zusammen mit der Vertreter-ID des aufrufenden Komponentenvertreters in einer Liste im Prozessvertreter abgelegt.

Aufruf eines Prozessdienstes

Wird ein Prozessdienst aufgerufen, so wird durch die angelegte Liste eine Zuordnung der Vertreter-ID zu einer Objekt-ID gemacht und die Interaktion über die prozessorientierte Schnittstelle mit den übermittelten Parametern dem entsprechenden Objekt übermittelt. Erfolgt eine Antwort auf die Interaktion, so wird diese von dem Objekt, auf die diese Antwort wirkt an den Prozessvertreter über die prozessorientierte Schnittstelle übermittelt, welcher sie über die abgelegte Liste an den entsprechenden Komponentenvertreter weitergibt.

3.3.3.4 Simulationsvertreter

Um das Konzept der prozessorientierten Interaktion zu ermöglichen müssen sowohl ein Prozessvertreter konzipiert als auch die Komponentenvertreter angepasst werden.

Prozessvertreter

Der Prozessvertreter muss die Dienstaufrufe der Komponentenvertreter an die entsprechenden Objekte in der Prozesssimulation weiterleiten. Um sie an das richtige Objekt weiterzuleiten nutzt er die Liste, die die Objekt-IDs den Vertreter-IDs zuordnet.

Zusätzlich muss er die Prozessdienste anbieten. Die angebotenen Prozessdienste werden bei der Anbindung einer Prozesssimulation an einen Prozessvertreter ermittelt, wobei jedem Prozess ein Dienst zugeordnet ist. Die Ermittlung erfolgt über eine im Modell enthaltene Spezifikation der im Modell enthaltenen Prozesse. Diese Spezifikation ist eine Anforderung an die Modellierung der Prozessmodelle, ohne die das Dienstekonzept nicht funktioniert. Wie diese Spezifikation aussieht, hängt von den Möglichkeiten des verwendeten Simulationstools ab. Sie kann beispielsweise in Form einer Initialisierungsnachricht, einer Initialisierungsdatei oder durch Kommentare im Modell erfolgen. Da diese Spezifikation simulationstoolspezifisch ist, muss der simulationstoolspezifische Teil der Schnittstelle auch hierfür angepasst werden.

Somit benötigt der Prozessvertreter sowohl eine Liste des Mappings von Vertreter-IDs mit Objekt-IDs als auch eine Liste der angebotenen Prozessdienste, siehe Abbildung 21.

Diese Prozessdienste muss er bei seinem Eintritt nicht an das Adressregister mitteilen, da sie im Fall der prozessorientierten Interaktion direkt von den Komponentenvertretern angefragt werden, da diese die Dienste ihren Schnittstellen zuordnen müssen.

Komponentenvertreter

Die Komponentenvertreter müssen die prozessorientierten Interaktionen von den Komponentensimulationen an die Prozessvertreter weiterleiten, hierzu nutzen sie die Dienste der Prozessvertreter. Da sie jedoch durch die Kapselung kein Wissen über die von ihnen vertretene Simulation haben, wissen sie nicht, bei welchem Output der Simulation welcher Dienst aufgerufen werden muss.

Daher muss ein Mapping der Ausgaben auf Prozessdienste erfolgen. Dies erfolgt beim Anbinden einer Komponentensimulation an ihren Komponentenvertreter. Dieser fordert zuerst die Liste der möglichen Interaktionsdienste bei den Prozessvertretern an. Damit diese Prozessdienste Ausgaben des Simulationstools zuordenbar sind, müssen die einzelnen Ausgaben voneinander getrennt sein, was am einfachsten über getrennte Schnittstellen möglich ist. In dem Fall kann jeder Schnittstelle ein Dienst zugeordnet werden. Sind bei einem Simulationstool nicht mehrere Schnittstellen möglich, so kann ein Modell eines derartigen Simulationstools nur über einen manuellen Aufwand eingebunden werden. In dieser Arbeit wird diese Möglichkeit nicht betrachtet, sondern es wird vorausgesetzt, dass die verwendeten Simulationstools trennbare Schnittstellen haben. Werden mehrere Prozesssimulationen verwendet für unterschiedliche Prozesse, welche aber alle beispielsweise Bewegungsdaten benötigen, so ist es auch möglich einer Schnittstelle mehrere Dienste, in diesem Fall von unterschiedlichen Prozesssimulationen, zuzuweisen. Dies wird aber ebenfalls nicht in dieser Arbeit berücksichtigt.

Um die tatsächliche Zuordnung der Dienste zu den Schnittstellen durchzuführen, bedarf es ebenfalls einer Spezifikation des Modells, welche besagt, welche Schnittstelle welchen Prozess abbildet. Allerdings kann sich hier das Problem ergeben, dass diese Spezifikation nicht die gleiche Wortwahl wie die Spezifikation der Prozesse des Prozessmodells verwendet, so kann beispielsweise der gleiche Prozess einmal mit „messe-Temperatur“ und einmal mit „measure_Temp“ beschrieben werden. Da das Problem der unterschiedlichen Wortwahl nicht ohne Weiteres zu beheben ist und die Hersteller der Komponentenmodelle nicht auf einen Standard zur Spezifikation der Prozesse gezwungen werden können, ist es hier am sinnvollsten auf eine manuelle Zuordnung der Dienste zu den Schnittstellen auszuweichen.

Der Aufwand hierfür ist vernachlässigbar, da die meisten Modelle nur sehr wenige Prozesse verwenden. Durch diese Zuordnung erfolgt nur die Einbindung und keine Änderung der vorhandenen Modelle oder der Interaktionen bzw. deren Modellierung. Somit entsteht hier kein Widerspruch zu den Anforderungen A1 und A2.

Nach diesem Mapping erweitert der Komponentenvertreter eine Liste, die die Schnittstellen mit den dazugehörigen Diensten verbindet, wodurch er die Ausgaben der Komponentensimulationen durch die entsprechenden Dienste weiterleiten kann, siehe Abbildung 21.

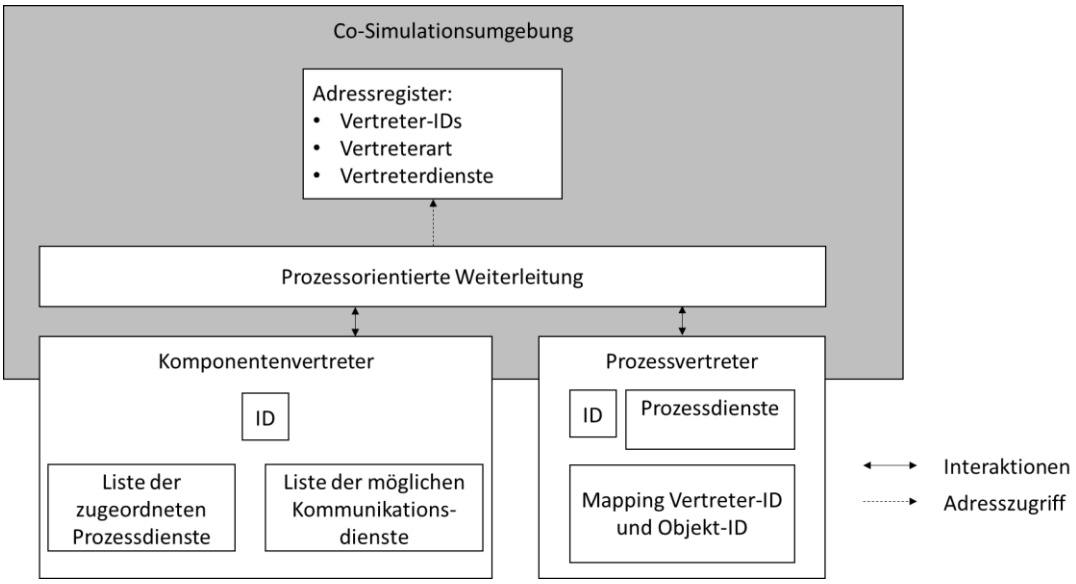


Abbildung 21: Co-Simulationsumgebung mit Prozessvertreter

3.3.3.5 Schnittstellen zwischen Simulationsvertreter und Simulation

Prozessorientierte Schnittstellen einer Prozesssimulation und einer Komponentensimulation unterscheiden sich leicht.

Prozessorientierte Schnittstelle der Prozesssimulation

Die prozessorientierte Schnittstelle der Prozesssimulation muss sowohl die Interaktionen an die Prozesssimulation weiterleiten und die Antworten auf diese Interaktionen zurückleiten als auch ein Mapping zwischen Vertreter-ID und Objekt-ID durchführen (Abbildung 22 Ein- und Ausgangsknoten).

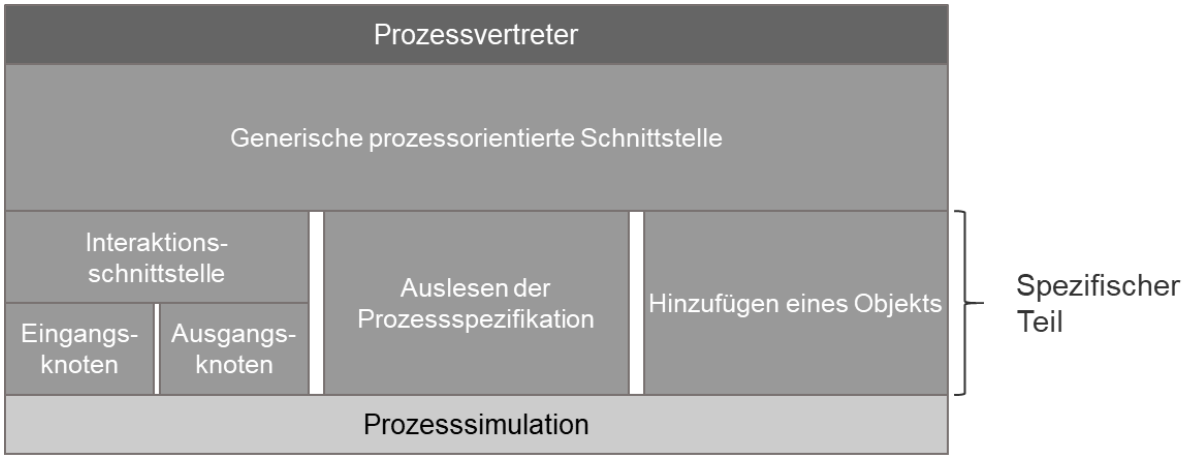


Abbildung 22: Prozessorientierte Schnittstelle einer Prozesssimulation

Zudem muss sie das Erstellen der Liste der Prozessdienste ermöglichen und somit die Spezifikation der Prozesse aus dem Modell auslesen. Auch das Hinzufügen eines Objekts muss über diese Schnittstelle getriggert werden können (Abbildung 22).

Prozessorientierte Schnittstelle der Komponentensimulation

Die prozessorientierte Schnittstelle der Komponentensimulation muss die Interaktionen an den Komponentenvertreter weiterleiten, wobei eine Trennung der unterschiedlichen Prozesse erfolgen muss (siehe Abbildung 23 Schnittstelle Prozess 1 und Schnittstelle Prozess 2). Zudem muss beim Einbinden einer Komponente ein Auslesen von deren Abmessungen bzw. Geometrie aus einem Geometrie-Modell erfolgen und an die Prozesssimulation weitergeleitet werden (Abbildung 23).

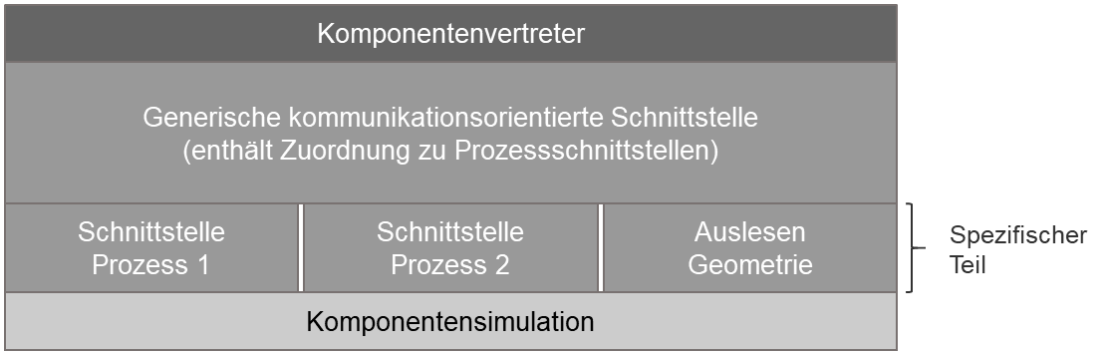


Abbildung 23: Prozessorientierte Schnittstelle einer Komponentensimulation

3.3.3.6 Verwendung mehrerer Prozesssimulationen

Da durch das Konzept die parallele Verwendung mehrerer Prozesssimulationen zulässig ist, bedarf es einer eindeutigen Identifikation der Dienste. Wenn bei jedem Dienst der zugehörige Simulationsvertreter hinterlegt wird, ist eine eindeutige Zuordnung der einzelnen Schnittstellen der Komponentensimulationen zu den Prozessdiensten möglich.

3.3.4 Anforderungen an die Teilsimulationen und ihre Simulationstools

Durch das interaktionsorientierte Konzept ergeben sich einige Anforderungen an die Simulationstools und an die Simulationen, um das Konzept der kommunikations- und prozessorientierten Interaktionen zu ermöglichen. Diese unterteilen sich in Anforderungen an die Komponentensimulationen, an die Kommunikationssimulationen und an die Prozesssimulationen.

Anforderungen an die Komponentensimulationen

Anforderungen durch die kommunikationsorientierte Interaktion:

- Da zur kommunikationsorientierten Interaktion die Spezifikation einer Kommunikationstechnologie genutzt wird, müssen das Simulationstool und die Modellierung der Komponente in der Lage sein, Nachrichten gemäß dieser Spezifikation zu generieren und interpretieren.

- Es muss eine Instanziierungsnachricht erzeugt werden können.

Anforderungen durch die prozessorientierte Interaktion:

- Die Abmessungen und Geometrie der Komponente müssen ebenfalls modelliert sein und an die Prozesssimulation durch den Instanziierungsdienst übermittelt werden.
- Nutzt die Komponente verschiedene Prozesse, so müssen diese über getrennte Schnittstellen abrufbar sein, um eine Zuordnung zu den Diensten zu ermöglichen.
- Die Komponentensimulationen dürfen keine physikalischen Prozesse enthalten, die andere Komponenten beeinflussen, diese müssen in der Prozesssimulation modelliert sein.

Diese Anforderungen schränken die Auswahl der Simulationstools nicht sonderlich stark ein. Die größte Einschränkung ist, dass in dem Simulationstool die Spezifikation der Kommunikationstechnologie berücksichtigt sein muss. Da bei der Simulation eines IoT-Systems die Kommunikation ebenfalls simuliert werden muss, ist diese Einschränkung erforderlich. Die Anforderung der getrennten Schnittstelle für die prozessorientierten Interaktionen könnte ebenfalls die Simulationstoolauswahl einschränken, allerdings hat eine Untersuchung im Rahmen dieser Arbeit gezeigt, dass gängige Simulationstools eigentlich immer mehrere Ausgänge anbieten, die für die getrennten Schnittstellen geeignet sind.

Anforderungen an die Kommunikationssimulationen

Folgende Anforderungen werden an die Kommunikationssimulation gestellt:

- Die Kommunikationssimulation muss ebenfalls die Spezifikation der Kommunikationstechnologie umsetzen. Dies sollte immer der Fall sein, da eine spezialisierte Kommunikationssimulation die Kommunikationstechnologie simuliert.
- Es müssen Knoten zur Laufzeit durch den Kommunikationsvertreter hinzugefügt werden können, diese Anforderung ist für „Plug-and-Simulate“ unumgänglich. Dies muss über eine Instanziierungsnachricht geschehen, welche simulationstoolspezifisch sein kann.
- Ist das Anlegen eines Knoten erfolgreich, so muss dies quittiert werden können.
- Die Knoten müssen durch den Kommunikationsvertreter eindeutig identifizierbar sein und es müssen Nachrichten einem Knoten zugewiesen werden können. Dies ist meist möglich, wenn Knoten von außen (durch den Kommunikationsvertreter) in die Simulation eingefügt werden können.
- Werden unterschiedliche Kommunikationstechnologien in einer Simulation simuliert, müssen für diese getrennten Schnittstellen verwendet werden. Sollten nicht getrennte Schnittstellen vorhanden sein, so kann die Simulation in mehrere Simulationen aufgeteilt werden.

Anforderungen an die Prozesssimulationen

Folgende Anforderungen werden an die Prozesssimulation gestellt:

- Es müssen Objekte zur Laufzeit durch den Prozessvertreter hinzugefügt werden können, diese Anforderung ist für „Plug-and-Simulate“ unumgänglich. Hierbei müssen sowohl Position und Ausrichtung des Objekts als auch dessen Geometrie übermittelt werden.
- Die Objekte müssen durch den Prozessvertreter eindeutig identifizierbar sein und er muss ihnen Interaktionen zuordnen können.
- Die in der Prozesssimulation angebotenen Dienste müssen in einer vom Prozessvertreter lesbaren Spezifikation beschrieben sein. Die Bereitstellung der Spezifikation kann simulationstoolspezifisch sein.

Die Anforderungen an die Simulationstools für die Kommunikationssimulationen und Prozesssimulationen schränken die Simulationstoolauswahl etwas ein, da diese jedoch in der Regel nicht von einem Komponentenhersteller geliefert werden, können sie durch den Nutzer frei gewählt werden, wodurch es zu keiner wirklichen Einschränkung der Nutzbarkeit des vorgestellten Co-Simulationskonzepts kommt.

3.4 Synchronisation der Co-Simulation

In einer Co-Simulation sind mehrere Teilsimulationen zu einer Gesamtsimulation gekoppelt. Da davon auszugehen ist, dass manche Teilsimulationen deutlich schneller ablaufen als andere, kann es aufgrund der daraus resultierenden unterschiedlichen Simulationszeiten zu Kausalitätsfehlern kommen.

Ein Kausalitätsfehler tritt dann auf, wenn eine Teilsimulation eine Nachricht von einer anderen Teilsimulation empfängt, die allerdings in der Vergangenheit der empfangenden Teilsimulation liegt, da diese Teilsimulation deutlich weiter in der Simulationszeit fortgeschritten ist als die sendende Teilsimulation. Dies soll an folgendem Beispiel verdeutlicht werden:

In der Realität sind seit dem Start der Co-Simulation 10 Sekunden vergangen. Teilsimulation A hat in dieser Zeit 30 Sekunden simuliert, Teilsimulation B hingegen nur 3 Sekunden. Nun sendet Teilsimulation B zum Zeitpunkt t = 10s eine Nachricht an Teilsimulation A. Da diese aber schon deutlich weiter in der Simulationszeit fortgeschritten ist, erhält sie diese Nachricht 27 Sekunden zu spät. Da diese Nachricht aber Teilsimulation A beeinflusst, wurden die letzten 27 Sekunden falsch simuliert. Werden solche Kausalitätsfehler nicht behandelt, ergibt sich am Ende ein falsches Ergebnis der Gesamtsimulation. Dieser Sachverhalt wird in Abbildung 24 verdeutlicht.

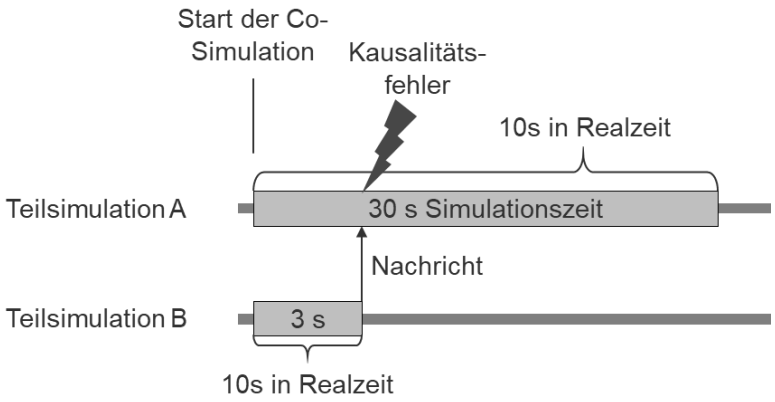


Abbildung 24: Kausalitätsfehler

Mit einer Synchronisation der Co-Simulation können solche Fehler vermieden oder behandelt werden. Da es nicht Ziel dieser Arbeit ist, ein neues Synchronisationskonzept zu entwickeln, werden im Folgenden verschiedene Synchronisationskonzepte vorgestellt und auf ihre Tauglichkeit für das oben beschriebene Konzept hin untersucht und bewertet. Anschließend wird das in dieser Arbeit für die Co-Simulation verwendete Synchronisationskonzept präsentiert.

3.4.1 Synchronisationsansätze

3.4.1.1 Zeitfortschritt in einer Co-Simulation

Bevor die Synchronisation in der Co-Simulation realisiert werden kann, muss untersucht werden, wie in Simulatoren die logische Zeit während der Simulation fortschreitet. In [177] werden drei Möglichkeiten des Zeitfortschritts in einer Simulation vorgestellt:

- Event Driven Time Advancing: Die Simulation ist eine diskrete Eventsimulation und die Simulationsschritte sind in Events unterteilt. In der Simulation werden diese Events nacheinander in der richtigen Reihenfolge ausgeführt. Diese Reihenfolge wird meist aus Zeitstempeln abgeleitet, die zu den Events gehören.

- Time Stepped Time Advancing: Die Simulation kann eine kontinuierliche Simulation oder eine diskrete Eventsimulation sein, die Simulation ist aber in Zeitschritte unterteilt. Diese Zeitschritte können entweder eine feste oder flexible Dauer haben, und alle Berechnungen für einen Zeitschritt müssen von allen Simulationen beendet werden, bevor der nächste Zeitschritt erreicht wird.

- Wall Clock Driven Time Advancing: Die Simulation kann eine kontinuierliche Simulation oder eine diskrete Eventsimulation sein. Der Zeitfortschritt hängt nicht nur von der Ausführungsgeschwindigkeit der Simulation ab, sondern auch von einer externen Wall Clock, die kontinuierlich läuft.

Abhängig von den verwendeten zeitlichen Fortschrittsmechanismen kann die Synchronisation einer Co-Simulation vereinfacht werden. Je weniger Zeitfortschrittsmethoden in einer Co-Simulation auftreten, desto einfacher kann das Synchronisationskonzept gestaltet werden. Um der oben erwähnten Heterogenität der IoT-Systeme gerecht zu werden, wird eine große Vielfalt an Simulationstools eingesetzt, weshalb dies nicht garantiert werden kann und alle zeitschreitenden Mechanismen für die Synchronisation berücksichtigt werden müssen.

Unter dieser Voraussetzung werden im Folgenden existierende Synchronisationskonzepte untersucht.

Traditionell werden Synchronisationsmethoden in konservative und optimistische Synchronisation unterteilt [140].

3.4.1.2 Konservative Synchronisation

Konservative Mechanismen garantieren, dass bei einer diskreten Eventsimulation kein Event außer der Reihe behandelt wird und bei einer kontinuierlichen Simulation keine Nachricht verzögert oder zu spät empfangen wird. Daher regelt entweder eine zentrale Instanz bei kontinuierlichen Simulationen den Fortgang der Simulationen und lässt einen Fortgang nur dann zu, wenn alle aktuellen Nachrichten empfangen und alle aktuellen Ereignisse verarbeitet wurden (Abbildung 25), oder es muss bei der diskreten Event Simulation ein Mechanismus implementiert werden, der garantiert, dass alle Ereignisse in der richtigen Reihenfolge zugestellt werden. Es existieren mehrere Mechanismen und Algorithmen für eine konservative Synchronisation, die sich in die Methoden "Synchroner Betrieb", " Deadlock-Vermeidung", "Deadlock-Erkennung und Wiederherstellung" und "konservative Zeitfenster" [178] einteilen lassen. Allen diesen Methoden ist gemeinsam, dass jedes Ereignis und jede Nachricht einen Zeitstempel benötigt, um die Vermeidung von Kausalitätsfehlern zu gewährleisten.

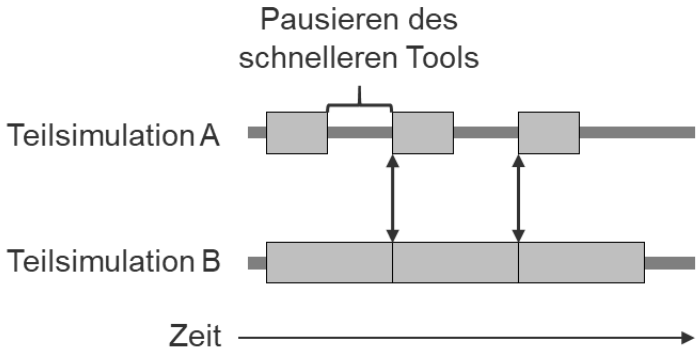


Abbildung 25: Konservative Synchronisation (Synchroner Betrieb)

3.4.1.3 Optimistische Synchronisation

Optimistische Synchronisationsalgorithmen erlauben Kausalitätsfehler und haben die Fähigkeit, diese zu erkennen. Für die Erkennung von Kausalitätsfehlern können Zeitstempel verwendet

werden, es gibt aber auch andere Möglichkeiten, diese zu erkennen. Wenn ein Kausalitätsfehler in einer Simulation erkannt wurde, muss die Simulation zurückgespult werden, d.h. alle vorhergehenden Simulationsergebnisse müssen rückgängig gemacht werden, bis der Kausalitätsfehler behoben ist. Vor dem Auftreten eines Kausalitätsfehlers sind die Simulationen einer Co-Simulation nicht synchronisiert und laufen daher unabhängig voneinander ab und werden nur im Falle eines Kausalitätsfehlers synchronisiert. Um eine optimistische Synchronisation zu realisieren, müssen alle bisherigen Eingaben der Simulation gespeichert werden, da sie im Falle eines Zurückspulens wieder benötigt werden. Zusätzlich muss ein Mechanismus zum Rückruf vorhergehender Ausgaben installiert werden, da im Falle eines Kausalitätsfehlers auch diese Ausgaben falsch sein können. Einer der bekanntesten optimistischen Synchronisationsalgorithmen ist der "Time Wrap Algorithmus" von Jefferson [179].

In Abbildung 26 wird eine optimistische Synchronisation dargestellt. Teilsimulation A läuft schneller als Teilsimulation B, kann aber so lange laufen, bis ein Kausalitätsfehler auftritt. In diesem Beispiel wäre das der Zeitpunkte zu dem eigentlich Event 2 hätte bei Teilsimulation A eintreffen müssen (gestrichelt). Dies ist aber erst später bekannt, nachdem schon ein weiterer Teil von Teilsimulation A weitergelaufen ist. Daher muss Teilsimulation A bis zu dem Zwischenergebnis, zu dem Event 2 hätte eintreffen müssen, zurückgespult werden und von dort an nochmals laufen.

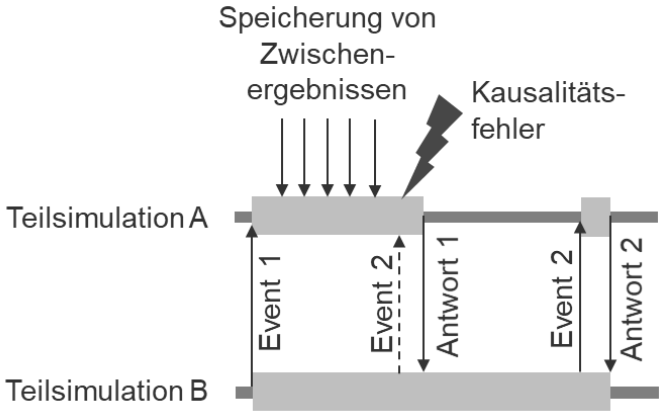


Abbildung 26: Optimistische Synchronisation

Ein Vorteil der optimistischen Synchronisationsmechanismen ist, dass die Simulationszeit stark reduziert werden kann, da die einzelnen Simulationen alle in ihrer eigenen Zeit ablaufen können und nicht regelmäßig auf eine langsamere Simulation warten müssen. Im besten Fall kann sogar die kürzest mögliche Ausführungszeit für die Co-Simulation erreicht werden. Die Ressourcenauslastung ist jedoch höher als bei konservativen Synchronisationsmethoden, da nicht alle berechneten Ergebnisse verwertbar sind.

3.4.2 Synchronisation bei Co-Simulationsstandards

Zusätzlich wurden die in den Co-Simulationsstandards FMI und HLA verwendeten Synchronisationskonzepte untersucht und auf eine Tauglichkeit zur Verwendung in der Co-Simulation dieser Arbeit hin untersucht. Diese Untersuchungen erfolgten, um die Praxistauglichkeit der oben beschriebenen Synchronisationskonzepte zu erörtern und um zu prüfen, ob ein ausgereiftes Synchronisationskonzept auf die Co-Simulation dieser Arbeit übertragbar ist.

3.4.2.1 Synchronisation bei FMI

Der Simulations-Master in FMI ist sowohl für den Datenaustausch zwischen den Teilsimulationen als auch für die Synchronisation der Teilsimulationen verantwortlich. Die am häufigsten verwendeten Synchronisationsmethoden bei FMI sind konservative Methoden, genauer gesagt "Synchroner Betrieb"-Methoden, so dass der Datenaustausch zwischen den Simulationen zu diskreten Zeitpunkten erfolgt. Diese diskreten Punkte werden als Kommunikationspunkte bezeichnet und alle Simulations-Slaves stoppen ihre Simulation an jedem Kommunikationspunkt. Dann sammelt der Simulations-Master alle Ausgänge der Simulations-Slaves und verteilt sie als Eingänge an die korrespondierenden Simulations-Slaves. An jedem Kommunikationspunkt stellt der Simulationsmaster auch die aktuelle Simulationszeit für jede Simulation und die Größe des nächsten Simulationsschritts, also die Zeit bis zum nächsten Kommunikationspunkt zur Verfügung. Diese Schrittgrößen können entweder konstant sein oder für jeden Simulationsschritt variieren [173].

3.4.2.2 Synchronisation bei HLA

Die Run Time Infrastructure (RTI) koordiniert den Datenaustausch und die Synchronisation der Co-Simulation bei HLA und ist vergleichbar mit einem Simulations-Master, ist aber selbst keine Simulation [180].

Für HLA gibt es sowohl konservative (zeitschritt- und eventbasierter Zeitfortschritt) als auch optimistische Synchronisationsmethoden. Um an der Synchronisation teilzunehmen, müssen die Föderate Time Stamp Orders (TSO) senden, die einen Zeitstempel enthalten, wann die Nachricht gesendet wurde. Zusätzlich können diese Nachrichten einen Lookahead-Wert enthalten, d.h. die Zeit, in der ein Föderat keine TSO-Nachrichten sendet, beginnend mit dem Zeitpunkt, an dem die letzte TSO-Nachricht gesendet wurde. Durch das Senden eines Lookahead-Wertes kann garantiert werden, dass in einer bestimmten Zeitspanne keine anderen Nachrichten gesendet werden und die anderen Föderate ihre Simulation für diesen Zeitraum problemlos ausführen können. Wenn ein Föderat einen zeitbasierten Zeitfortschritt hat, muss er bei der RTI einen Time Advance Request stellen, der erst dann gewährt wird, wenn die RTI geprüft hat, ob die Bedingungen für einen Zeitfortschritt erfüllt sind. Bei eventbasiertem Zeitfortschritt funktioniert es ähnlich, nur, dass ein

"Next Message Request" an die RTI gestellt werden muss. Wird ein optimistischer Ansatz verwendet, müssen die Methoden der Message Retraction angewendet werden. Wenn ein Föderat eine Nachricht mit einer geringeren logischen Zeit als seine eigene Simulationszeit erhält, muss er eine "Request Retraction"-Meldung an die anderen Föderate senden, um die falsch gesendeten Nachrichten zurückzuziehen. Die Erkennung von Kausalitätsfehlern muss jedoch von dem Föderat selbst gehandhabt werden [174].

3.4.3 Bewertung der Synchronisationsansätze

Da es nicht Ziel dieser Arbeit ist, einen neuen Synchronisationsalgorithmus zu entwickeln und eine möglichst hohe Performance zu erzielen, sondern die prinzipielle Machbarkeit einer dynamischen Co-Simulation zu zeigen, muss auch kein hochperformanter Synchronisationsalgorithmus gewählt werden, da ein solcher normalerweise komplizierter und somit schwieriger umzusetzen ist.

Sowohl FMI als auch teilweise HLA verwenden eine konservative Synchronisation, welche sich somit bei Co-Simulationsstandards bewährt hat und einfach zu realisieren ist. HLA verwendet in manchen RTIs komplexere Algorithmen, welche für diese Arbeit aber nicht weiter berücksichtigt werden.

Prinzipiell lässt sich der Algorithmus von FMI auf die präsentierte dynamische Co-Simulation übertragen, allerdings bedarf es in der präsentierten dynamischen Co-Simulation nicht der Einschränkung, dass nur zu Synchronisationszeitpunkten ein Datenaustausch stattfinden darf, da die Synchronisation komplett vom Datenaustausch entkoppelt ist.

3.4.4 Synchronisationskonzept

Es wird der Grundansatz von FMI gewählt und dahingehend abgeändert, dass zu jedem Zeitpunkt ein Datenaustausch zwischen den Teilsimulationen möglich ist und somit die Synchronisation komplett vom Datenaustausch entkoppelt ist. Gesteuert wird die Synchronisation von einem zentralen Taktgeber. Diese Methode erlaubt auch die Einbindung von allen Simulationstools, die eine der oben beschriebenen Zeitfortschrittsmethoden verwenden.

In den folgenden zwei Unterkapitel werden die für diese Methode benötigten Bestandteile, Taktgeber und Synchronisationsschnittstelle beschrieben, anschließend werden die daraus entstehenden Anforderungen an die Simulationstools erläutert.

3.4.5 Taktgeber

Wie schon in 3.1.2 erwähnt, wird die Synchronisation durch einen zentralen Taktgeber gesteuert. Die Synchronisation erfolgt in sich wiederholenden Synchronisationszyklen. Ein

Synchronisationszyklus beginnt damit, dass der Taktgeber an alle Simulationsvertreter den Start eines neuen Synchronisationszyklus und die Dauer des zu simulierenden Zeitschritts übermittelt. Zu diesem Zeitpunkt sind alle Teilsimulationen (Komponenten- und Interaktionssimulationen) pausiert. Die Simulationsvertreter starten daraufhin ihre Simulationen. Sobald eine Simulation den aktuellen Zeitschritt simuliert hat, pausiert sie und teilt dem Simulationsvertreter dies mit, dieser meldet dem Taktgeber, dass seine Simulation wieder pausiert ist. Sobald der Taktgeber von allen Teilsimulationen diese Rückmeldung erhalten hat, beginnt er den nächsten Synchronisationszyklus.

Die Länge des zu simulierenden Zeitschritts wird durch den Taktgeber bestimmt, in dem er von den Simulationsvertretern erfasst, was die minimal möglichen Zeitschritte des jeweiligen Simulationstools sind. Die Länge des Zeitschritts muss so klein gewählt werden, dass Kausalitätsfehler ausgeschlossen werden können. Dies hängt sehr stark vom simulierten Szenario ab. Kausalitätsfehler müssen auch dann ausgeschlossen werden können, wenn eine Nachricht während dem Pausieren einer Teilsimulation von einer langsameren Teilsimulation eintrifft. Es darf dann nicht zu einem Kausalitätsfehler kommen, wenn so eine Nachricht erst im nächsten Simulationsschritt abgearbeitet wird, obwohl sie theoretisch zum vorherigen zeitschritt gehört hätte. Dieser wird dann als Zeitschritt gewählt, auch wenn ein größerer Zeitschritt performanter wäre, was aber nicht das Ziel dieser Arbeit ist.

Die Simulationsvertreter pausieren und starten ihre Simulationen durch die Synchronisationsschnittstelle.

3.4.6 Synchronisationsschnittstelle

Über die Synchronisationsschnittstelle müssen die Simulationen pausiert und wieder gestartet werden. Zusätzlich wird ein Auslesen des minimalst möglichen, ausführbaren Zeitschritts benötigt (Abbildung 27). Dies geschieht beim Anbinden der Simulation an den Simulationsvertreter.

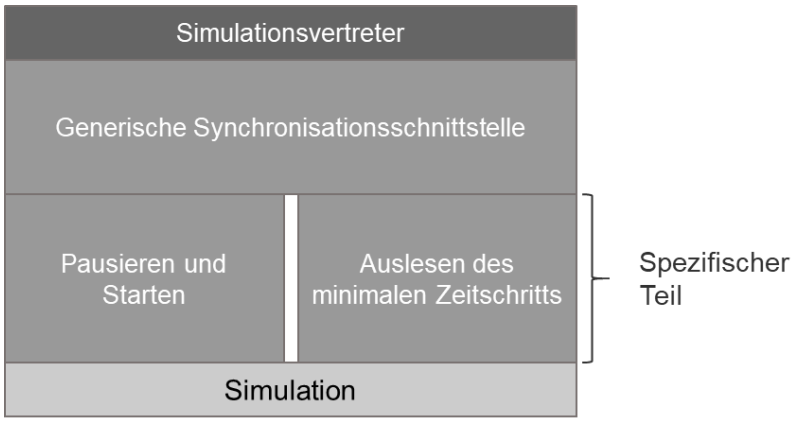


Abbildung 27: Synchronisationsschnittstelle

3.4.7 Anforderungen an die Teilsimulationen und ihre Simulationstools

Alle Simulationstools müssen in der Lage sein, ihre Simulationen zu festen Zeitpunkten zu stoppen, und müssen daher in der Lage sein, einen kontinuierlichen Fluss der Simulationszeit zu haben, oder, falls dies nicht der Fall ist, müssen sie in der Lage sein, Ereignisse zu unterteilen, die länger als die eingestellte Simulationsschrittweite dauern. Wenn das Konzept um die Möglichkeit erweitert wird, dass nicht alle Simulationstools an jedem Kommunikationspunkt angehalten werden müssen, sondern einige nur an einigen Kommunikationspunkten teilnehmen, würde die Anzahl der nutzbaren Simulationstools steigen.

Zusätzlich müssen diese Simulationstools dem Simulationsvertreter mitteilen können, welches ihre minimal möglichen Zeitschritte sind.

3.4.8 Erweiterung der Synchronisation

Da es nicht Ziel dieser Arbeit ist, einen neuen Synchronisationsalgorithmus zu entwickeln und eine möglichst hohe Performance zu erzielen, sondern die prinzipielle Machbarkeit einer dynamischen Co-Simulation zu zeigen, wurde ein möglichst simpler, einfach umzusetzender Synchronisationsalgorithmus gewählt. Dieser Algorithmus kann beliebig erweitert und in seiner Mächtigkeit gesteigert werden. Dies führt zum einen zu einer höheren Performanz, zum anderen wird die Anzahl der einsetzbaren Simulationstools gesteigert, da durch eine Erweiterung durch einen optimistischen Synchronisationsalgorithmus auch Simulationstools verwendet werden können, welche nicht pausiert aber zurückgespult werden können.

Diese Erweiterungen sind allerdings nicht Gegenstand dieser Arbeit.

In diesem Kapitel wurde das Konzept für eine dynamische Co-Simulation vorgestellt, in dem die Komponenten durch Simulationsvertreter und die Interaktionen zwischen ihnen durch Interaktionssimulationen gekapselt sind, um ein dynamisches Eintreten zu ermöglichen. Darüber hinaus wurde ein Konzept für einen Datenaustausch unter Berücksichtigung eines dynamischen Eintretens sowie die Interaktionssimulationen vorgestellt. Abschließend wurden unterschiedliche Synchronisationsansätze diskutiert und ein konservativer Ansatz für das vorgestellte Konzept angepasst.

Nach der Methode der Design Science wurden mehre Iterationen über das Konzept durchgeführt. Hierbei wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie unterstützt. Dies gilt insbesondere für die einzelnen Teilkonzepte, beispielsweise bei der Synchronisierung gab es Iterationen darüber, welches Synchronisationskonzept den Anforderungen der Co-Simulation als auch Anwendungsfällen aus der Industrie genügt, ohne die Komplexität unnötig zu erhöhen. Auch die Unterteilung in kommunikations- und prozessorientierte Interaktionen erfolgt im Laufe der einzelnen Iterationen. Nach jeder Iteration wurde eine Kontrolle anhand der

aufgestellten Anforderungen durchgeführt sowie auf die Tauglichkeit des Konzepts hinsichtlich eines zukünftigen Einsatzes in der Industrie über die genannten Diskussionen mit Experten aus der Industrie.

4 Realisierung der Co-Simulation als Framework

Um den Entwicklungsaufwand für die Implementierung des Konzepts so gering wie möglich zu halten, wurde ein Framework konzipiert. Ziel der Realisierung des Frameworks ist es, dem Nutzer eine problemlose Verwendung und Nutzung der Co-Simulationsumgebung zu ermöglichen und den Entwicklungsaufwand für das Einbinden von eigenen Simulationstools und Simulationen zu minimieren.

Jede Simulation und somit jede Co-Simulation hat eigene Simulationsziele und somit eigene Modelle bzw. eine eigene Zusammenstellung dieser Modelle. Dennoch können gewisse Teile für jede Co-Simulation wiederverwendet werden, insbesondere ist die Co-Simulationsumgebung für jede Co-Simulation gleich. Auch die Schnittstellen zur Anbindung der Simulationstools können immer wieder verwendet werden, falls in unterschiedlichen Co-Simulationen die gleichen Simulationstools verwendet werden. Die Interaktionssimulationen können ebenfalls in der Regel wiederverwendet werden, da oftmals die gleichen Kommunikationstechnologien eingesetzt werden und gleiche oder ähnliche physikalische Prozesse ablaufen.

Das Framework muss drei Hauptteile anbieten:

- Ein Framework der Co-Simulationsumgebung
- Ein Konzept zur Erstellung der Schnittstellen zu den Simulationstools
- Eine Anleitung zur Erstellung der Interaktionssimulationen

Die Realisierungen dieser drei Hauptteile sind unabhängig voneinander. Im Folgenden werden zu jedem Hauptbestandteil verschiedene Realisierungsmöglichkeiten diskutiert, wobei es prinzipiell möglich ist die Realisierungsmöglichkeit eines Hauptteils auszutauschen ohne die anderen Hauptteile anpassen zu müssen.

In Kapitel 4.1 wird sowohl aus Sicht des Frameworkbetreibers als auch aus Sicht des Frameworknutzers diskutiert, mit welcher Technologie bzw. mit welchem Paradigma die Co-Simulationsumgebung zu realisieren ist. Da die Simulationsvertreter sehr eng mit der Co-Simulationsumgebung zusammenhängen wird beides zusammen betrachtet und realisiert.

Das Konzept zur Erstellung der Schnittstellen zu den einzelnen Simulationstools wird in Kapitel 4.2 vorgestellt. Dieses Konzept dient der Erreichung einer höheren Wiederverwendbarkeit bei der Einbindung von Simulationstools. Hierbei wird berücksichtigt, dass Nutzer des Co-Simulationsframeworks, Betreiber des Co-Simulationsframeworks und Simulationstoolanbieter die Simulationstools möglichst einfach an die Co-Simulation anbinden können.

Abschließend wird in Kapitel 4.3 eine Anleitung zur Realisierung der Interaktionssimulationen vorgestellt. Hierbei werden ebenfalls vor allem Aspekte der Wiederverwendbarkeit diskutiert, um eine Modellbibliothek zu ermöglichen.

Die Implementierung des Frameworks sowie der einzelnen Teilsimulationen wird in Kapitel 5 vorgestellt.

4.1 Konzept des Frameworks der Co-Simulationsumgebung und der Simulationsvertreter

Die grundlegende Entscheidung bei der Realisierung der Co-Simulationsumgebung ist die Entscheidung für die Technologie oder das Paradigma, mit welcher sie umgesetzt wird. Grundlage für diese Entscheidung sind die Anforderungen, die aus dem präsentierten Konzept entstehen, sowie die Anforderungen, die durch die späteren Nutzer sowie den späteren Betreiber des Co-Simulationsframeworks entstehen.

Da die Simulationsvertreter sehr eng mit der Co-Simulationsumgebung verbunden sind, ist es sinnvoll eine gemeinsame Realisierung der Co-Simulationsumgebung und der Simulationsvertreter anzustreben, da dies eine höhere Performanz und Stabilität sowie einen geringeren Implementierungsaufwand verspricht.

4.1.1 Anforderungen an das Co-Simulationsumgebungsframework

4.1.1.1 Anforderungen aus dem Konzept

Die Anforderungen, die sich aus dem Konzept ergeben, wurden schon in Kapitel 3 beschrieben und ergeben sich hauptsächlich aus den Anforderungen an die Konzeption aus Kapitel 1.3. Um einen einfacheren Vergleich der Realisierungsmöglichkeiten zu ermöglichen, werden sie hier noch einmal kurz zusammengefasst.

Realisierungsanforderung RA1: Es muss möglich sein, dass die Simulationsvertreter zur Laufzeit in die Co-Simulationsumgebung eintreten, ohne dass diese pausiert werden muss (A1, A2, A4).

Realisierungsanforderung RA2: Es muss möglich sein, dass die Simulationsvertreter zur Laufzeit aus der Co-Simulationsumgebung austreten, ohne dass diese pausiert werden muss (A1, A2, A4).

Realisierungsanforderung RA3: Es muss möglich sein, dass die Simulationsvertreter Nachrichten über eine Dienststruktur austauschen (A2).

Realisierungsanforderung RA4: Es muss eine Art Adressregister vorhanden sein, in dem die Simulationsvertreter sowie die von ihnen angebotenen Dienste gefunden werden können (A2). Über dieses Adressregister müssen die Simulationsvertreter eindeutig identifizierbar sein (A2).

Realisierungsanforderung RA5: Es muss unterschiedliche Arten von Simulationsvertretern geben können (Komponentenvertreter, Kommunikationsvertreter und Prozessvertreter) (A3).

4.1.1.2 Kriterien gestellt durch Nutzer und Betreiber

Zusätzlich zu den funktionalen Anforderungen, die sich aus dem Konzept ergeben, ergeben sich Kriterien, welche durch Nutzer und Betreiber des Co-Simulationsframeworks gestellt werden, die die Wahl der Realisierungsmöglichkeit beeinflussen.

Kriterien des Nutzers

Plattformunabhängigkeit: Aus Sicht des Nutzers ist die Plattformunabhängigkeit wünschenswert, um das Framework auf unterschiedlichen Systemen betreiben zu können.

Plattformübergreifender Einsatz: Zum einen können Simulationen oftmals sehr ressourcenauslastend sein, weshalb ein PC nicht ausreicht, um alle Teilsimulationen performant auszuführen, zum anderen können unterschiedliche Simulationstools unterschiedliche Systemanforderungen (z.B. bzgl. des Betriebssystems) haben. In beiden Fällen ist es erforderlich, dass die Teilsimulationen auf unterschiedlichen PCs ausgeführt werden.

Kriterien des Betreibers

Integration in den Digitalen Zwilling: Der Digitale Zwilling eines IoT-Systems ist oftmals aus den Digitalen Zwillingen der einzelnen Komponenten aufgebaut, weshalb die Simulation des Digitalen Zwillings des Gesamtsystems eine Co-Simulation der Digitalen Zwillinge der einzelnen Komponenten ist. Da der Digitale Zwilling auch während des Betriebs eingesetzt wird, bietet sich hier eine dynamische „Plug-and-Simulate“-fähige Co-Simulation an. Diese Möglichkeit wird in [27] erörtert und betont, dass die Co-Simulation ein wichtiger Bestandteil des Digitalen Zwillings ist. Somit ist eine einfache Integration der Co-Simulation in ein Digitales-Zwillings-Konzept wünschenswert.

Beherrschung der Technologie: Falls ein Betreiber eine Technologie bzw. ein Paradigma schon beherrscht, trägt dies zu einer effizienteren Entwicklung und einem stabileren Betrieb bei. Wird nur die Co-Simulation betrachtet, spielt es keine Rolle, welche Technologie bzw. Paradigma für die Umsetzung verwendet wird, so lange die Realisierungsanforderungen RA1 - RA5, die sich aus dem Konzept ergeben, erfüllt sind.

Dies ist keine abschließende Aufstellung der Anforderungen. Die in dieser Arbeit vorgestellte Implementierung ist eine mögliche Realisierung des in Kapitel 3 präsentierten Konzepts. Aus Nutzer oder Betreibersicht könnte die Anforderungsliste geändert oder ergänzt werden. Die gewählten Anforderungen haben einen direkten Einfluss auf die in Kapitel 4.1.3 getroffene Entscheidung.

4.1.2 Realisierungsmöglichkeiten

Im Folgenden werden einige Realisierungsmöglichkeiten mit unterschiedlichen Technologien und Paradigmen ohne Anspruch auf Vollständigkeit vorgestellt. Prinzipiell ist es auch möglich andere Technologien und Paradigmen einzusetzen, so lange sie die Realisierungsanforderungen RA1 - RA5 erfüllen.

Die hier vorgestellten Technologien und Paradigmen erfüllen diese Anforderungen. Ausgewählt wurden OSGi, eine Realisierung mit einem serviceorientierten Ansatz und ein Agentenansatz, da diese sich, wie der Stand der Forschung in Kapitel 2.4 zeigt, bei Co-Simulationen bewähren.

4.1.2.1 OSGi

In OSGi können die sogenannten Bundles zur Laufzeit ein- bzw. austreten, zudem können sie Daten untereinander austauschen, dies kann auch in einer diensteorientierten Weise erfolgen. Es ist zudem problemlos möglich, OSGi durch ein Adressregister zu erweitern und den Bundles Eigenschaften zuzuweisen, wie die Art des Simulationsvertreters. Somit erfüllt OSGi alle Realisierungsanforderungen, siehe zum Vergleich Kapitel 2.4.7.

Um das Konzept mit OSGi zu realisieren bildet OSGi die Co-Simulationsumgebung, die Simulationsvertreter werden durch OSGi-Bundles realisiert, welche sich in OSGi ein- und ausklinken können, siehe Abbildung 28. Das Adressregister und der Taktgeber werden ebenfalls als Bundle realisiert und hinzugefügt, siehe ebenfalls Abbildung 28.

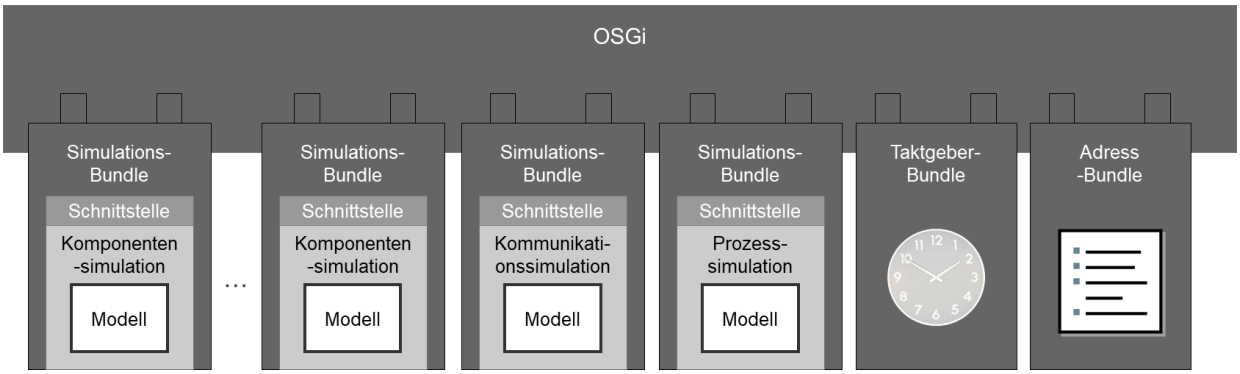


Abbildung 28: Realisierung mit OSGi

4.1.2.2 Serviceorientierung

Eine weitere Möglichkeit die Co-Simulationsumgebung zu realisieren ist der Einsatz einer serviceorientierten Architektur mit einem serviceorientierten Protokoll, wie beispielsweise OPC UA. Im Folgenden wird OPC UA als Beispieltechnologie genommen, ist aber durch andere serviceorientierte Architekturen ersetzbar, solange diese ebenfalls die Realisierungsanforderungen erfüllen.

Bei serviceorientierten Architekturen ist es prinzipiell möglich, dass Teilnehmer zur Laufzeit eintreten oder austreten, die Kommunikation zwischen ihnen erfolgt über Dienste, welche meist in einer Art Adressregister gelistet sind, wie beispielsweise der Discovery-Server in OPC UA. Dieses Adressregister kann als Adressregister der Co-Simulation genutzt werden. Den einzelnen Servern und Clients können zusätzliche Eigenschaften zugeordnet werden um die Simulationsvertreterart zu unterscheiden. Somit sind alle Realisierungsanforderungen erfüllt, siehe zum Vergleich 2.4.6.

Die Co-Simulationsumgebung kann durch einen Server realisiert werden, der eine Verbindung zu einem Client in jedem der Simulationsvertreter hat. Um eine bidirektionale Kommunikation zu ermöglichen, benötigt jeder Simulationsvertreter einen Server und die Co-Simulationsumgebung einen Client, siehe Abbildung 29. Der Taktgeber wird ebenfalls als Server-Client-Paar realisiert, siehe ebenfalls Abbildung 29. Um die Datenaustauschschleife zu schließen, besitzt das Server-Client-Paar der Co-Simulationsumgebung ebenfalls eine Verbindung, siehe Abbildung 29.

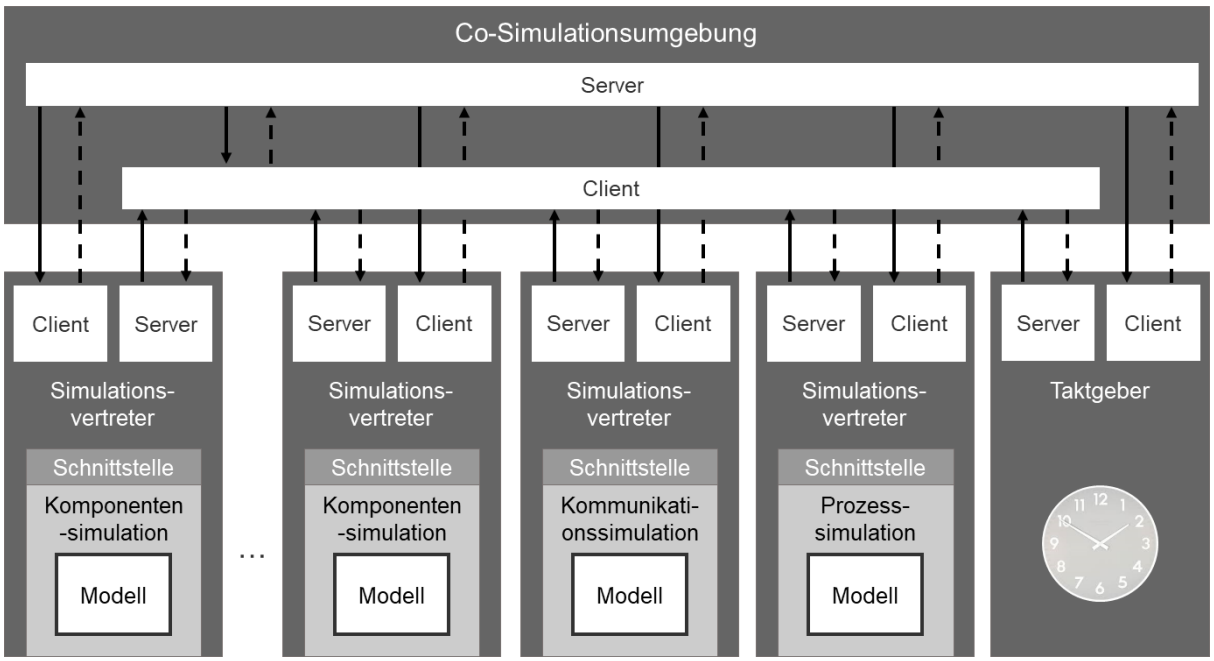


Abbildung 29: Realisierung über eine serviceorientierte Architektur

4.1.2.3 Softwareagenten

Auch Softwareagenten bieten die Möglichkeit zur Laufzeit in ein Agentensystem einzutreten bzw. auszutreten. Die Kommunikation kann ebenfalls über Dienste erfolgen und das Adressregister wird vom Agenten-Management-System zusammen mit dem Directory Facilitator gebildet. Agenten können auch verschiedene Rollen einnehmen, wodurch RA 5 abgedeckt ist. Somit werden auch alle Realisierungsanforderungen von Agenten erfüllt, siehe zum Vergleich 2.4.8.

Bei einer Realisierung des Frameworks mit Agenten, wird die Co-Simulationsumgebung durch die Agentenumgebung inklusive Agenten-Management-System und Directory Facilitator

realisiert und die Simulationsvertreter durch Agenten umgesetzt. Auch der Taktgeber wird durch einen Agenten realisiert (Abbildung 30).

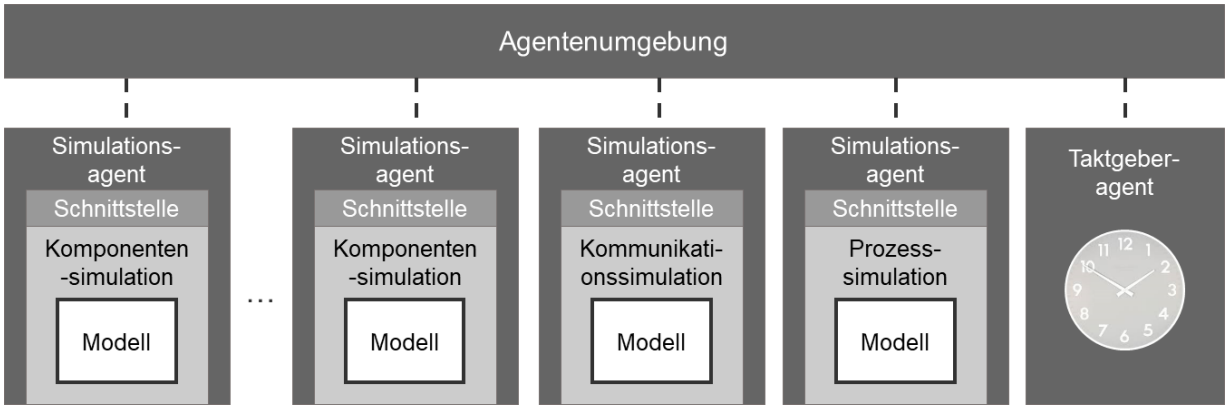


Abbildung 30: Realisierung durch Agenten

4.1.3 Diskussion der Realisierungsmöglichkeiten

Da alle vorgestellten Möglichkeiten alle Realisierungsanforderungen erfüllen, sind die von den Nutzern und Betreibern des Frameworks aufgestellten Kriterien entscheidend.

Plattformunabhängigkeit: Alle drei Möglichkeiten können plattformunabhängig sein. OSGi ist Java-basiert und auch Agenten können mit JADE Java-basiert realisiert werden, wodurch sie eine gute Plattformunabhängigkeit erhalten. OPC UA, was ein Beispiel für eine serviceorientierte Architektur ist, ist ebenfalls plattformunabhängig.

Plattformübergreifender Einsatz: Auch alle drei Möglichkeiten können über mehrere Plattformen hinweg kommunizieren. Bei OSGi müssen kleine Erweiterungen vorgenommen werden, Agenten, durch ihre Eigenschaft der Mobilität (siehe 2.4.8.1) und auch OPC UA durch seine Server-Client-Architektur können problemlos über mehrere Plattformen hinweg betrieben werden.

Integration in den Digitalen Zwilling: Für die Bewertung der Integrationsmöglichkeiten in den Digitalen Zwilling wird die Architektur des Intelligenten Digitalen Zwillings aus [27] verwendet. OSGi und serviceorientierte Architekturen bieten grundsätzlich nur eine geeignete Kommunikationsbasis für den Digitalen Zwilling. So sind sie zwar problemlos in einen Digitalen Zwilling integrierbar, unterstützen diese Integration aber nicht allzu sehr. OPC UA, als Beispiel für eine serviceorientierte Architektur unterstützt diese Integration durch sein Informationsmodell etwas stärker als OSGi. Agenten unterstützen diese Integration noch etwas stärker durch ihre Eigenschaften der Autonomie, Handlungsspielraum, Zielorientierung und Anpassungsfähigkeit. Diese Fähigkeiten zielen alle auf ein autonomes Handeln des Agenten zum Erreichen eines Ziels ab. Dieses Vorgehen wird ebenfalls beim Digitalen Zwilling angestrebt, so dass man beim Intelligenten Digitalen Zwilling aus [27] ebenfalls von einer Art Agent sprechen kann, was für diese Anforderung die Verwendung von Agenten nahelegt.

Beherrschung der Technologie: Dieses sehr subjektive Kriterium fällt ebenfalls zu Gunsten der Agenten aus, da an dem Institut, an dem diese Arbeit entstanden ist, deutlich mehr Vorwissen über Agenten als über OSGi oder serviceorientierte Architekturen vorhanden ist. Da es sich bei der hier zu treffenden Entscheidung nur um eine Realisierungsentscheidung handelt und nicht um eine konzeptionelle Entscheidung und alle vorgestellten Möglichkeiten die Anforderungen, die sich aus dem Konzept ergeben erfüllen, kann die Beherrschung der Technologie als zusätzliches Entscheidungskriterium angewandt werden.

Tabelle 8 fasst die Bewertung der einzelnen Kriterien zusammen.

Tabelle 8: Vergleich der Realisierungsmöglichkeiten

Kriterium / **Ansatz**	**Plattform-unabhängigkeit**	**Plattform-übergreifender Einsatz**	**Integration in den Digitalen Zwilling**	**Beherrschung der Technologie**
OSGi	●	◕	◔	◑
Serviceorientierte Architekturen	●	●	◑	◑
Agenten	●	●	◕	●

● Erfüllt ◕ Weitestgehend Erfüllt ◑ Teilweise Erfüllt ◔ Prinzipiell Erfüllt

Aus dieser Aufstellung wird ersichtlich, dass Agenten die Kriterien aus Sicht der Nutzer und Betreiber des Frameworks im Rahmen dieser Arbeit am besten erfüllen. Daher wird das Framework in dieser Arbeit durch Agenten, wie in 4.1.2.3 präsentiert, realisiert.

4.2 Konzept zur Erstellung der Schnittstellen

4.2.1 Konzeption der Schnittstellen

Jedes Simulationstool, welches in das Co-Simulationsframework eingebunden werden soll, benötigt eine Implementierung der Schnittstellen für den Datenaustausch und die Synchronisation. Ziel bei der Implementierung ist es eine möglichst hohe Wiederverwendbarkeit und Übertragbarkeit zu erreichen.

In einem ersten Schritt wird die Wiederverwendbarkeit durch die Aufteilung in einen generischen und einen simulationstoolspezifischen Teil (siehe Konzeptbeschreibung in 3.2.3) ermöglicht. Im simulationstoolspezifischen Teil befinden sich nur noch die Codeteile, die sich nicht auf alle Simulationstools übertragen lassen. Dennoch ist es erstrebenswert zusätzlich Codeteile zu

identifizieren, die auf andere Simulationstools übertragen werden können. Um dies zu erreichen werden die von den Simulationstools bereitgestellten Schnittstellen untersucht und in Kategorien eingeteilt. Tabelle 9 zeigt die hierzu untersuchten Simulationstools und die Kategorien der Schnittstelle, die diese verwenden. Zusätzlich zu den Simulationstools wird noch eine Anbindung an FMI über FMUs untersucht, genauer beschrieben in 4.2.3.

Tabelle 9: Simulationstools und die von ihnen verwendeten Schnittstellen

Simulationstool	**Art der Schnittstelle**
MATLAB Simulink (Komponentensimulation)	Verwendung einer API (matlab.engine)
OpenModelica (Komponentensimulation)	Kommunikation über CORBA
AnyLogic (Komponentensimulation)	Kommunikation über Sockets
Unity 3D (Prozesssimulation)	Kommunikation über Sockets
OMNet++ (Kommunikationssimulation)	Kommunikation über Sockets
Anbindung an FMI über FMUs (Komponentensimulation)	Verwendung einer API (fmi4j)

Hierbei zeigt sich, dass es bei dieser Simulationstoolauswahl drei prinzipielle Kategorien von Schnittstellen verwendet werden:

- Simulationstooleigene APIs
- Kommunikation über Sockets
- Kommunikation über CORBA

Innerhalb dieser Kategorien können Programmteile wiederverwendet werden, wie die Funktionsprinzipien der einzelnen Schnittstellenarten zeigen.

Simulationstooleigene APIs

APIs (Application Programming Interface) sind dokumentierte Schnittstellen die von einem Simulationstool angeboten werden. Über die APIs läuft sowohl der Datenaustausch als auch die Synchronisation. Meistens wird der Zugriff auf diese Schnittstellen durch bereitgestellte Bibliotheken in einer oder mehreren Programmiersprachen erleichtert. Bei MATLAB und FMI ist dies beispielsweise in Java der Fall.

Da häufig diese Bibliotheken existieren, kann in der Regel mit geringem Aufwand eine schnelle Anbindung des Simulationstools vorgenommen werden. Da die Bibliotheken simulationstoolspezifisch sind, ist in dieser Kategorie der Wiederverwendungsgrad gering, dennoch sollte der Zugriff auf die APIs so weit wie möglich gekapselt werden.

Eine detaillierte Beschreibung des Zugriffs auf die APIs von MATLAB Simulink und FMI wird in den Kapiteln 5.2.2.1 und 5.2.2.4 gegeben.

Kommunikation über Sockets

Sockets sind standardisierte Kommunikationspunkte welche durch das Betriebssystem verwaltet werden. Sockets sind bidirektional und können über eine Kombination aus IP-Adresse und Portnummer adressiert werden. Somit ist es sowohl möglich zwischen einzelnen PCs Daten auszutauschen als auch innerhalb eines PCs Daten zwischen Programmen, denen Ports zugewiesen wurden, auszutauschen. Die Kommunikation zwischen Sockets läuft über ein standardisiertes Protokoll, wie beispielsweise TCP-IP ab.

Die Schnittstellen von Simulationstools, die über Sockets kommunizieren, verwenden meistens eine Server-Client-Architektur, wobei der Server Teil des Simulationstools ist und der Client über die Socket-Verbindung auf diesen Server zugreifen kann.

Da der Aufbau einer Socket-Verbindung meist ähnlich abläuft, kann hierfür ein Grundgerüst vorgegeben werden, für das zwar simulationstoolspezifische Anpassungen (z.B. Anpassungen von Verbindungsaufbauparametern wie Portnummer) vorgenommen werden müssen, jedoch von allen Simulationstools mit Socket-Kommunikation weitestgehend verwendet werden kann. Da meist ein Server Teil des Simulationstools ist, muss in der simulationstoolspezifischen Schnittstelle ein Client enthalten sein, dessen Verbindung zu dem generischen Teil der Schnittstelle einheitlich sein kann.

Somit kann der Rahmen der simulationstoolspezifischen Schnittstelle des Simulationsvertreters, der die Verbindung zum Client innerhalb der Schnittstelle herstellt, komplett wiederverwendet werden. Der Client sowie der Aufbau der Socket-Kommunikation können teilweise wiederverwendet werden, der simulationstoolseitige Server und der Aufbau der Socket-Kommunikation auf der Simulationstoolseite müssen hingegen für jedes Simulationstool neu angelegt werden, dies ist aber von dem Simulationstool vorimplementiert, weshalb nur eine Instanziierung erfolgen muss, siehe Abbildung 31.

Eine detaillierte Beschreibung der Verbindung zu den Simulationstools AnyLogic, OMNet++ und Unity 3D wird in Kapiteln 5.2.2.3, 5.2.3 und 5.2.4 gegeben.

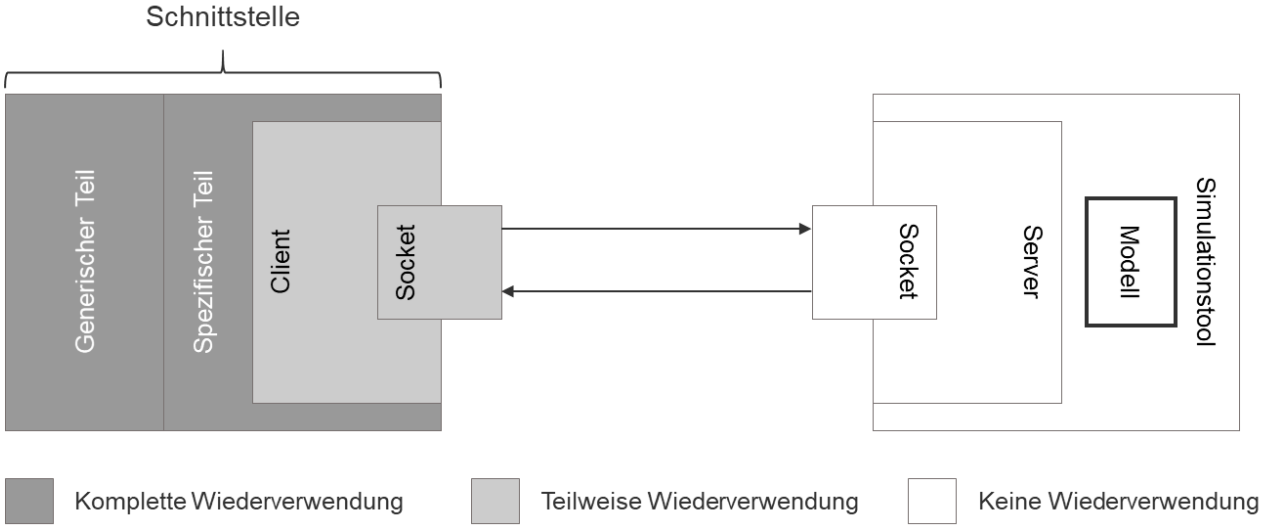


Abbildung 31: Kommunikation mit einem Simulationstool über Sockets

Kommunikation über CORBA

Die Kommunikation bei der Common Object Request Broker Architecture läuft ebenfalls über eine Server-Client-Architektur ab. Hierbei wird von dem Simulationstool ein Server instanziiert, mit dem Clients über einen Object Request Broker kommunizieren können. Der Object Request Broker bildet das Herzstück einer CORBA-Kommunikation und wird ebenfalls von dem Simulationstool bereitgestellt. Somit muss in der Schnittstelle des Simulationsvertreters ein Client erzeugt werden, der sich dann über den Object Request Broker mit dem Server im Simulationstool verbindet.

Dieses Erzeugen des Clients kann wie bei der Socket-Kommunikation gekapselt werden, sodass der Teil, der die Verbindung zum Client innerhalber der Schnittstelle herstellt, komplett wiederverwendet werden kann. Nur der Client selbst und dessen Erzeugung müssen teilweise angepasst werden. Auch der Object Request Broker sowie der Server im Simulationstool müssen nicht implementiert, sondern nur instanziiert werden, siehe Abbildung 32.

Eine detaillierte Beschreibung der Verbindung zu dem Simulationstool OpenModelica wird in Kapitel 5.2.2.2 gegeben.

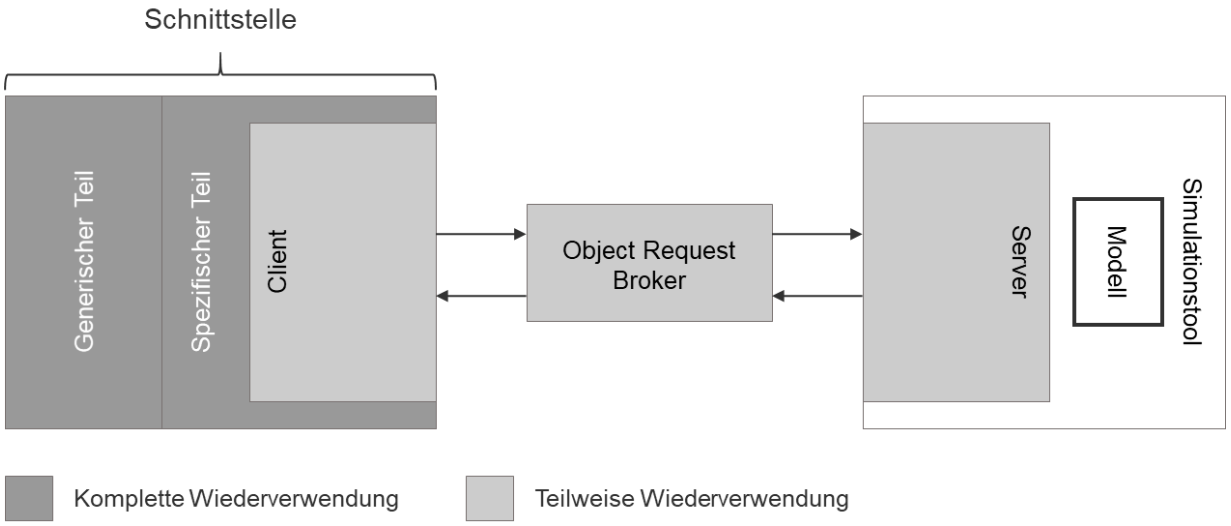


Abbildung 32: Kommunikation mit einem Simulationstool über CORBA

Diese Beispiele zeigen, dass es auch in den simulationstoolspezifischen Teilen der Schnittstelle des Simulationsvertreters Wiederverwendungspotential gibt. Hierzu können die einzelnen Simulationstools in Kategorien nach den von ihnen verwendeten Schnittstellen gegliedert werden. Da es aber nicht Ziel dieser Arbeit ist eine umfassende Anbindung von einer Vielzahl an Simulationstools zu implementieren, sondern die prinzipielle Machbarkeit einer „Plug-and-Simulate“-fähigen Co-Simulation nachzuweisen, wird hier nur das Vorgehen zum Erreichen einer höheren Wiederverwendbarkeit vorgestellt.

In Zukunft muss eine umfassendere Untersuchung an Simulationstools erfolgen und die einzelnen spezifischen Schnittstellen der unterschiedlichen Schnittstellenkategorien generisch vorimplementiert werden, um über generische Vorimplementierungen für Socket-Kommunikation und eine generische Vorimplementierung für eine Kommunikation über CORBA zu verfügen, welche für die einzelnen Simulationstools nur noch minimal angepasst werden müssen.

4.2.2 Befähigung von zusätzlichen Simulationstools

Eine Anforderung an die eingesetzten Simulationstools zur Simulation der Komponenten, die sich aus dem Konzept der Interaktionssimulation ergibt, ist, dass die Simulationstools und die sich darin befindlichen Modelle in der Lage sein müssen Nachrichten nach der Kommunikationsspezifikation zu erzeugen und zu verstehen. Dies ist allerdings bei einer Vielzahl an Simulationstools nicht der Fall, obwohl sie prinzipiell in der Lage wären IoT-Komponenten zu simulieren. Ein Beispiel hierfür ist MATLAB Simulink, mit dem Sensoren simuliert werden können aber nur Integer als Ausgangswerte zur Verfügung stehen, mit denen beispielsweise keine IP-Adressen dargestellt werden können.

Um dennoch diese Simulationstools an das Co-Simulationsframework anzubinden, ist es notwendig die Modellierung der Kommunikationsnachrichten, bzw. deren Formate in die

Simulationsvertreter bzw. deren Schnittstelle zu verlegen, da die Modellierung nicht im Simulationstool selbst erfolgen kann. Hierzu kann ein sogenannter Nachrichten-Mapper zwischen den spezifischen Teil und den generischen Teil der kommunikationsorientierten Schnittstelle geschoben werden, siehe Abbildung 33.

Dieser Nachrichten-Mapper wandelt die Ausgangsdaten der Simulation, die von dem simulationstoolspezifischen Teil der Schnittstelle abgegriffen werden, in Nachrichten um, die der Spezifikation der Kommunikationstechnologie entsprechen und gibt diese dann an den generischen Teil weiter. Somit erscheint es wieder für den Komponentenvertreter, als ob die komplette Modellierung im Simulationstool erfolgt wäre.

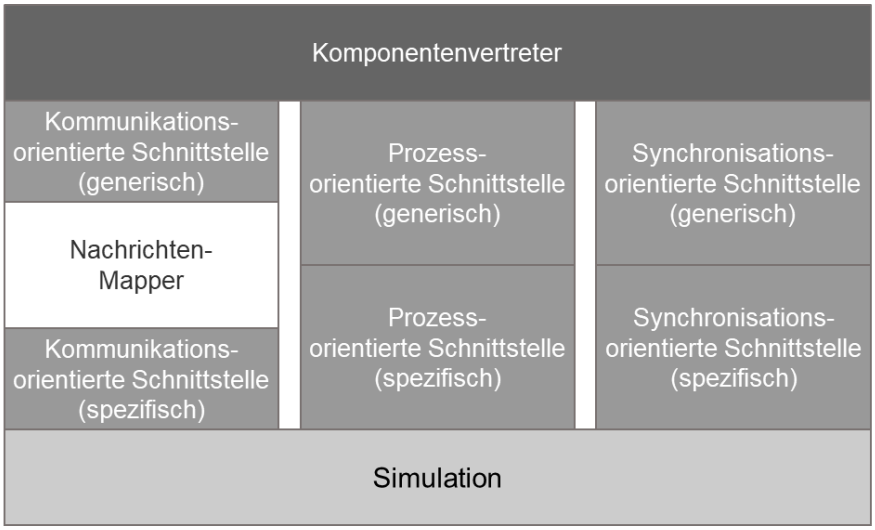


Abbildung 33: Befähigung von zusätzlichen Simulationstools durch Nachrichten-Mapping

Dieser Nachrichten-Mapper erfordert einen manuellen Modellierungsaufwand beim Eintritt einer Teilsimulation. Dies ist aber kein Widerspruch zu den Anforderungen A1 und A2, da es sich hierbei nur um eine Erweiterung zur Befähigung von zusätzlichen Simulationstools handelt. Lässt man diese Simulationstools außen vor, so gilt weiterhin die Anforderung, dass im Simulationstool die Nachrichten nach der Spezifikation modelliert werden und somit ist kein Modellierungsaufwand beim Eintritt einer Komponente nötig.

4.2.3 Anbindung an FMI

Zusätzlich zu einer Anbindung von normalen Simulationstools wird eine Anbindung, wie schon in 4.2.1 erwähnt, an FMI durch FMUs umgesetzt. Ziel hierbei ist es, eine möglichst breite Anbindung von Simulationstools zu ermöglichen. Da schon eine Vielzahl an Simulationstools FMI unterstützen, sollte mit dieser Anbindung versucht werden, durch die Umsetzung einer einzigen Schnittstelle die Anbindung einer großen Zahl von Simulationstools auf einmal abzudecken. Somit können alle Simulationstools angebunden werden, die in der Lage sind, FMUs zu erzeugen. Schematisch ist die Verbindung zu einer FMU in Abbildung 34 dargestellt.

Über die Schnittstelle wird zu der Functional Mock-up Unit eine Verbindung aufgebaut. Diese enthält sowohl das Modell als auch einen Solver um das Modell auszuführen. Die Verbindung wird über eine von FMI zur Verfügung gestellte API (fmi4j) realisiert. Eine detaillierte Beschreibung dieser Verbindung wird in Kapitel 5.2.2.4 gegeben.

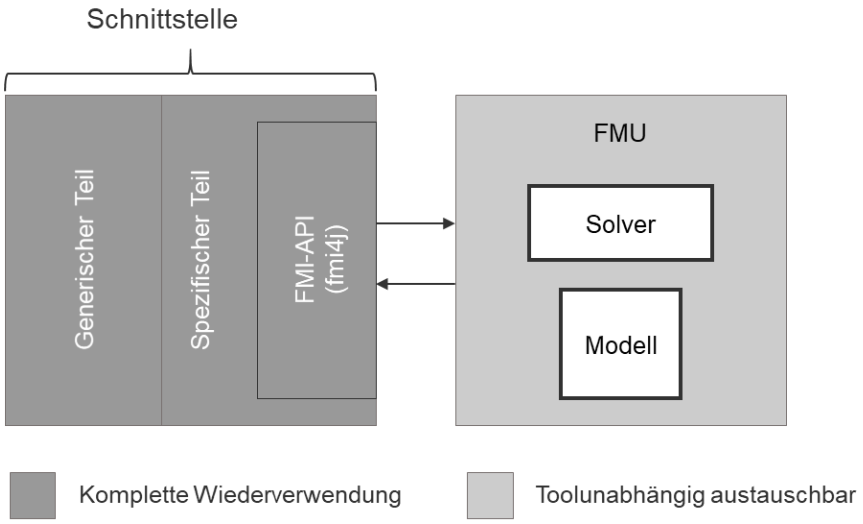


Abbildung 34: Anbindung von FMI über eine von FMI zur Verfügung gestellten API

Im Rahmen dieser Arbeit werden FMUs sowohl mit MATLAB Simulink als auch mit OpenModelica erstellt. Hierbei ist eine Implementierung der Schnittstelle für die FMUs beider Simulationstools ausreichend, womit gezeigt wird, dass FMUs aller FMI-kompatiblen Simulationstools integriert werden können. Bei MATLAB Simulink ist ein Nachrichten-Wrapper, wie in 4.2.2 vorgestellt, nötig, da MATLAB Simulink nicht in der Lage ist die Nachrichten spezifikationskonform zu erstellen.

4.3 Modellbibliothek zur Erstellung der Interaktionssimulationen

Auch bei den Interaktionssimulationen ist es wünschenswert, eine möglichst hohe Wiederverwendbarkeit der Modelle zu erreichen, um den nennenswerten Aufwand der Erstellung eines Interaktionsmodells zu vermeiden. Grundsätzlich können zwar in jedem IoT-System unterschiedliche Interaktionen auftreten, bezüglich der Kommunikation werden jedoch oftmals die gleichen Kommunikationstechnologien eingesetzt. Auch bei den prozessorientierten Interaktionen gibt es einige Prozesse, die in fast jedem IoT-System vorkommen, wie beispielsweise Reibung oder andere Kräfte zwischen zwei Komponenten. Daher kann zumindest ein Teil der Modelle wiederverwendet werden und muss nicht für jede Co-Simulation neu implementiert werden.

Um diese Wiederverwendbarkeit nutzbar zu machen, bietet sich eine Modellbibliothek an, in der sowohl Kommunikations- als auch Prozessmodelle hinterlegt sind. In den beiden folgenden

Unterkapiteln wird diskutiert, was bei der Erstellung der Modelle beachtet werden muss, um eine derartige Modellbibliothek zu ermöglichen.

4.3.1 Kommunikationsorientierte Interaktionssimulation

Der Nachrichtenaustausch mit kommunikationsorientierter Interaktionssimulation wird nach der Spezifikation der Kommunikationstechnologie modelliert. Dies hat den Vorteil, dass die Nachrichten für alle IoT-Systeme die gleiche Form haben und somit gleich behandelt werden können. Folglich ist es möglich für jede Kommunikationstechnologie eine Simulation zu erstellen, welche von unterschiedlichen Co-Simulationen verwendet werden kann.

Der Detailgrad der Modellierung und die simulierten Aspekte können unterschiedlich sein. Mit Aspekten sind hier beispielsweise Nachrichtenverluste, Störungen der Nachrichtenübertragung, Dämpfung bei der Nachrichtenübertragung und ähnliches gemeint. Um eine Wiederverwendung zu ermöglichen, sollen der Detaillierungsgrad und die Aspekte einstellbar und die Aspekte zu- bzw. abschaltbar sein.

Zusätzlich kann die räumliche Verteilung des IoT-Systems einen Einfluss auf die oben genannten Aspekte haben, hierzu können die Kommunikationssimulationen mit einer räumlichen Dimension ausgestattet werden, die ebenfalls parametrierbar ist.

Schwierig wird es, wenn die Kommunikation von der Bewegung der Komponenten im Raum abhängig ist, wie zum Beispiel bei RFID. In diesem Fall muss die Kommunikationssimulation mit der Prozesssimulation zusammengeschlossen werden oder die Bewegungsdaten der Komponentensimulationen müssen auch an die Kommunikationssimulation versendet werden. Prinzipiell ist eine derartige Simulation mit dem präsentierten Co-Simulationskonzept möglich, allerdings steigt hierbei der Modellierungsaufwand.

Somit ist es recht leicht möglich Kommunikationsmodelle in eine Modellbibliothek einzubinden, allerdings müssen gewisse Aspekte dieser Kommunikationsmodelle parametrierbar sein.

4.3.2 Prozessorientierte Interaktionssimulation

Bei prozessorientierten Interaktionen muss, wie in 3.3.3 begründet, die Spezifikation des Nachrichtenaustauschs durch die Prozesssimulation erfolgen. Dadurch wird es deutlich schwieriger eine Prozesssimulation auf mehrere Co-Simulationen zu übertragen, da deutlich mehr Freiheitsgrade in der Detaillierung der Modelle existieren. Zusätzlich gibt es eine größere Vielfalt an Prozessen, die modelliert werden können, was ebenfalls die Übertragbarkeit erschwert.

Daher kann versucht werden, Grundgerüste von Prozessmodellen anzubieten in denen elementare Prozesse enthalten sind, die von vielen Prozesssimulationen genutzt werden können, wie beispielsweise Bewegungen, Kollisionen zwischen Komponenten oder Temperaturausbreitung.

Diese Prozessmodelle können um weitere Prozesse erweitert werden. Hierbei ist darauf zu achten, dass die Prozesse modular modelliert sind, um einzelne Prozesse leicht abzuschalten oder zuzuschalten, sollten sie gebraucht oder nicht gebraucht werden. Zusätzlich sollten Einschränkungen beim Detaillierungsgrad vermieden werden, so dass es möglich ist auch Komponentenmodelle anzubinden, die einen geringeren Detaillierungsgrad aufweisen.

In diesem Kapitel wurden zuerst unterschiedliche Möglichkeiten zur Realisierung der Co-Simulationsumgebung sowie der Simulationsvertreter diskutiert und eine Entscheidung für die Umsetzung mit Softwareagenten getroffen. Danach wurde ein Konzept zur Erstellung der Schnittstellen vorgestellt, welches eine hohe Wiederverwendbarkeit vorsieht um den Aufwand zur Integration eines neuen Simulationstools zu minimieren. Anschließend wurde eine Modellbibliothek vorgestellt, ebenfalls mit dem Ziel eine hohe Wiederverwendbarkeit zu ermöglichen. Eine mögliche Erweiterung der dynamischen Co-Simulation um eine Hardware-in-the-Loop-Simulation wird im Anhang präsentiert.

Nach der Methode der Design Science wurden auch mehre Iterationen über die Realisierungsmöglichkeiten durchgeführt. Hierbei wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie unterstützt. Beispielsweise die Umsetzungsmöglichkeiten der Vertreter wurden in mehreren Iterationen erarbeitet und abschließend bewertet. Auch die detaillierte Ausarbeitung der Schnittstellen wurde über mehrere Iterationen erarbeitet. Dies geschah parallel zu der Verfeinerung des Konzepts, wodurch beispielsweise erst in einer späteren Iteration die Unterteilung in kommunikations- und prozessorientierte Schnittstelle eingefügt wurde. Nach jeder Iteration wurde eine Kontrolle anhand der aufgestellten Anforderungen durchgeführt sowie auf die Tauglichkeit der Realisierungen hinsichtlich eines zukünftigen Einsatzes in der Industrie über die genannten Diskussionen mit Experten aus der Industrie.

Die Implementierung des in Kapitel 3 beschriebenen Konzepts sowie der Realisierungsüberlegungen aus Kapitel 4 wird im folgenden Kapitel beschrieben.

5 Implementierung des Co-Simulationsframeworks

In diesem Kapitel wird ein Überblick über die tatsächliche Implementierung des in Kapitel 3 beschriebenen Konzeptes in Form eines Frameworks gegeben. Zuerst wird das Co-Simulationsframework mit seinen Simulationsvertretern, realisiert in Jadex, und anschließend werden die Implementierungen der einzelnen Schnittstellen, realisiert in Java beschrieben (Abbildung 35). Hierbei werden sowohl die Implementierungen der Schnittstellen zu den Simulationstools der Komponentensimulationen (MATLAB Simulink, OMNet++, AnyLogic, die Anbindung an FMI sowie eine Anbindung an eine reale Komponente) als auch die Anbindung der Kommunikationssimulation in OMNet++ und die Anbindung der Prozessinteraktion in Unity 3D (Abbildung 35). Danach wird eine kurze Beschreibung der Synchronisation gegeben und die Realisierung in den einzelnen Simulationstools dargestellt. Abschließend werden die Modellierungen der Interaktionssimulationen dargelegt. Die Modelle der einzelnen Komponenten werden in der Evaluierung in Zusammenhang mit dem Evaluationsszenario beschrieben.

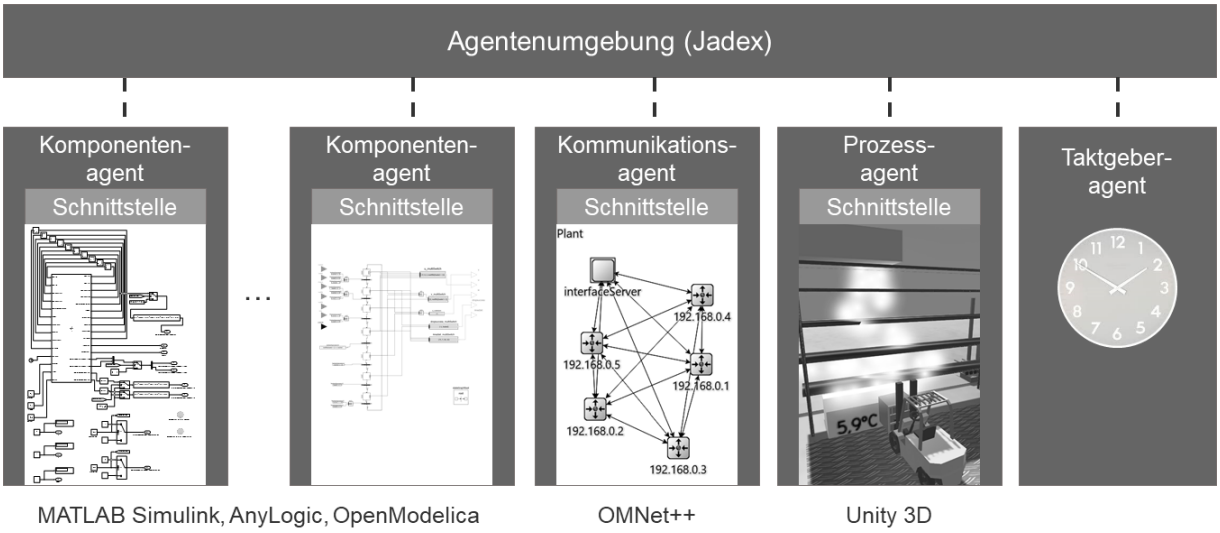


Abbildung 35: Übersicht über die Implementierung

Die meisten Teile der Implementierung wurden in zahlreichen in sich abgeschlossenen Arbeiten umgesetzt. Hier wird nur ein Überblick über diese Implementierungen gegeben, für eine detailliertere Beschreibung wird auf diese Arbeiten verwiesen.

5.1 Implementierung des Co-Simulationsframeworks

5.1.1 Co-Simulationsumgebung

In Kapitel 4.1.3 wurde die Entscheidung getroffen, das Framework durch ein Agentensystem zu realisieren. In der Implementierung wurde diese Entscheidung weiter auf BDI-Agenten eingeschränkt, da diese durch ihre Struktur einem Intelligenten Digitalen Zwillings nach [27] am

nächsten kommen. Zudem bieten sie eine gute Möglichkeit Dienste zu realisieren [181]. Eine Beschreibung des BDI-Konzepts findet sich in 2.4.8.1.

Außerdem wurde entschieden die Implementierung in Java durchzuführen, da es zum einen mit Jadex eine ausgereifte Bibliothek für Agentensystem gibt und zum anderen Java plattformunabhängig ist, wodurch die Forderung nach Plattformunabhängigkeit aus 4.1.1.2 erfüllt wird.

Die Co-Simulationsumgebung wird durch die Agent Platform in Jadex realisiert und muss die Nachrichtenweiterleitung, die „Plug-and-Simulate“-Fähigkeit sowie das Adressbuch beinhalten.

Die Bestandteile der Co-Simulationsumgebung werden wie folgt durch das Agentensystem realisiert:

- Die Nachrichtenweiterleitung wird durch Dienste, für welche ein in Jadex vorgefertigtes Interface verwendet wird, realisiert. Diese werden im Directory Facilitator angeboten, welcher ebenfalls von Jadex vorimplementiert ist. Der tatsächliche Datenaustausch erfolgt über den Message Transport Service.
- Die „Plug-and-Simulate“-Fähigkeit wird durch das Agent Management System ermöglicht, welches ebenfalls schon in Jadex enthalten ist.
- Das Adressbuch wird sowohl durch den Directory Facilitator (enthält die Dienste) als auch durch das Agent Management System (enthält die Agenten-IDs) realisiert

5.1.2 Simulationsvertreter

Die Simulationsvertreter werden als BDI-Agenten implementiert. Soll eine neue Teilsimulation an das Co-Simulationsframework angebunden werden, wird ein neuer Simulationsagent erzeugt, dieser wiederum erzeugt den generischen Teil der Schnittstelle. Der generische Teil der Schnittstelle erzeugt eine Benutzerabfrage, welches Modell eingebunden werden soll und in welchem Simulationstool dies ausgeführt wird. Aufgrund der Eingabe des Nutzers, wird die entsprechende spezifische Schnittstelle erzeugt.

Die Simulationsagenten bieten, wie im Konzept beschrieben, Dienste an, indem sie das vorimplementierte Interface von Jadex nutzen. Um das Mapping der Dienste zu den Schnittstellen der Simulationstools durchzuführen, erhalten sie eine Liste der Simulationstoolschnittstellen vom generischen Teil der Datenaustauschschnittstelle. Im Fall der Interaktionssimulationen generieren sie daraus die Liste der Dienste, die sie anbieten. Im Fall der Komponentensimulationen wird das Mapping manuell durchgeführt.

5.1.3 Benutzeroberfläche

Zusätzlich wird im Rahmen dieser Arbeit eine Benutzeroberfläche zu dem Co-Simulationsframework implementiert, mit der einzelne Teilsimulationen hinzugefügt und entfernt werden können und die Vorgänge in der Co-Simulation veranschaulicht werden können. Sie ist in Abbildung 36 zu sehen. Es werden die einzelnen Teilsimulationen und die Nachrichten, die zwischen ihnen ausgetauscht werden, bildlich dargestellt (rechter Teil), links unten sind Ereignisse, wie Ein- und Austritte von Teilsimulationen dargestellt, links in der Mitte werden die versendeten Nachrichten in der Co-Simulation aufgelistet und können sortiert und gefiltert werden und links oben kann die Co-Simulation über „Play“, „Pause“ und „Step“ gesteuert werden. Neue Teilsimulationen können über den Reiter „Agent“ hinzugefügt werden. Das sich hierbei öffnende Fenster wird in 5.2.1 beschrieben.

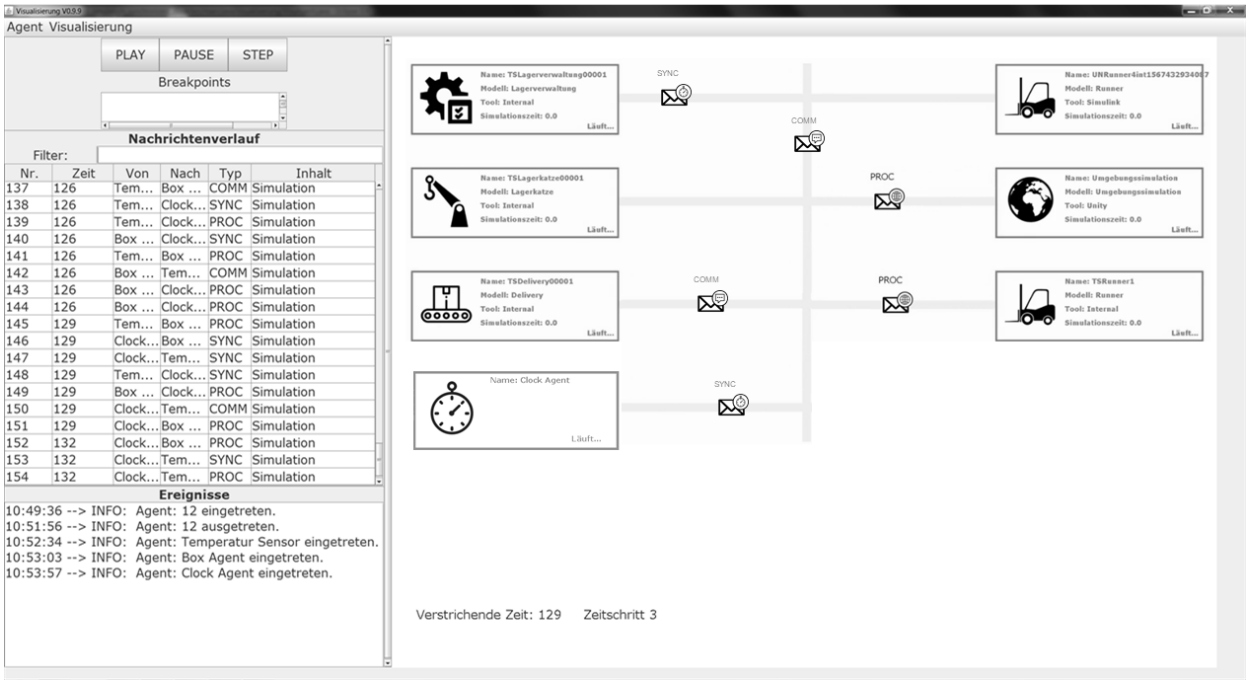


Abbildung 36: Benutzeroberfläche des Frameworks

Mit Hilfe der Benutzeroberfläche können die Interaktionen zwischen den Teilsimulationen nachvollzogen werden. Sie unterstützt somit die Analyse der Co-Simulation.

5.2 Implementierung der Schnittstellen

In diesem Unterkapitel werden die einzelnen spezifischen Schnittstellen zu den unterschiedlichen Simulationstools kurz vorgestellt, eine ausführlichere Beschreibung wird jeweils referenziert.

5.2.1 Generische Schnittstellen

Soll eine neue Teilsimulation eingebunden werden, wird ein neuer Agent instanziiert, der wiederum den generischen Teil der Schnittstelle instanziiert. Hierbei öffnet sich das Fenster in Abbildung 37, über welches der Nutzer eingeben muss, welches Modell er einbinden möchte und in welcher Simulation (Simulationstool und Instanz des Simulationstools) dieses ausgeführt wird. Zudem muss ein Name vergeben werden. Wird der Name „Umgebungssimulation“ vergeben, so wird ein Prozessagent erzeugt, eine prozessorientierte Schnittstelle für eine Prozesssimulation und eine Synchronisationsschnittstelle, wird der Name „Kommunikationssimulation“ vergeben, wird ein Kommunikationsagent sowie eine kommunikationsorientierte Schnittstelle für eine Kommunikationssimulation und eine Synchronisationsschnittstelle erzeugt. In allen anderen Fällen werden ein Komponentenagent sowie die drei Schnittstellen für Komponentenagenten (kommunikationsorientiert, prozessorientiert und Synchronisationsschnittstelle) erzeugt.

Abbildung 37: Benutzeroberfläche zum Einbinden einer Teilsimulation

Nachdem ausgewählt wurde, welches Modell in welchem Simulationstool eingebunden werden soll, werden die entsprechenden spezifischen Schnittstellen instanziiert.

5.2.2 Schnittstellen zu Komponentensimulationen

5.2.2.1 Schnittstelle zu MATLAB Simulink

Um eine Verbindung zu einer MATLAB-Simulink-Simulation aufzubauen, wird die von MATLAB bereitgestellte API matlab.engine.shareEngine verwendet. Diese API wird in MATLAB Simulink ausgeführt, wodurch die Session, in der diese Simulation ausgeführt wird, freigegeben wird. Anschließend kann auf diese Session von Java aus zugegriffen werden. Hierzu wird ein Objekt vom Typ Matlab.Engine erzeugt, über das die einzelnen Variablen in MATLAB Simulink gesetzt und ausgelesen werden können. Verwendet wurde die von MATLAB bereitgestellt Bibliothek com.mathworks.engine.

5.2.2.2 Schnittstelle zu OpenModelica

Um eine Verbindung mit OpenModelica herzustellen, wurde eine Kommunikation über einen CORBA-Server erstellt. Hierbei erzeugt OpenModelica einen CORBA-Server, auf den mit einem CORBA-Client von Java aus zugegriffen werden kann. Verwendet wurde die Bibliothek omc.corba.

5.2.2.3 Schnittstelle zu AnyLogic

Da die Personal Learning Edition von AnyLogic verwendet wurde, konnte nicht auf die API von AnyLogic zugegriffen werden. Daher musste die Kommunikation über eine Socket-Verbindung realisiert werden. In AnyLogic ist es allerdings nicht möglich, eine dauerhafte Verbindung mit einem externen Programm aufrecht zu erhalten und gleichzeitig zu simulieren. Daher muss die Socket-Verbindung unterbrochen werden, wenn simuliert werden soll, was zur Folge hat, dass in der Personal Learning Edition von AnyLogic zyklisch zwischen den beiden Zuständen Simulieren und Datenaustausch gewechselt werden muss.

- Simulieren: In diesem Zustand wird der aktuelle Zeitschritt simuliert, bei Beendigung des Zeitschritts wird dieser Zustand verlassen.
- Datenaustausch: Bei Eintritt in diesen Zustand wird ein Server erzeugt und eine Socket-Verbindung aufgebaut, über die der Datenaustausch abgewickelt wird. Anschließend wird, wenn der Befehl vom Taktgeber kommt, die Socket-Verbindung wieder geschlossen und der Zustand wird verlassen.

Java-seitig wird von einem Client kontinuierlich versucht eine Verbindung zu dem Server in AnyLogic aufzubauen. Sobald AnyLogic in den Zustand Datenaustausch wechselt, ist dies erfolgreich und ein Datenaustausch kann stattfinden.

Da ein Datenaustausch mit AnyLogic nicht kontinuierlich, sondern nur zu bestimmten Zeitpunkten zwischen zwei Zeitschritten der Co-Simulation möglich ist, ist seine Eignung für eine Co-Simulation einschränkt.

5.2.2.4 Schnittstelle zu FMI

Um eine Anbindung an eine Simulation über FMI zu ermöglichen wird eine FMU für eine Co-Simulation benötigt, die von einem FMI-fähigen Simulationstool exportiert werden muss. Diese FMU kann dann über die FMI-API fmi4j, einer Open-Source-Bibliothek in Java eingebunden werden. Hierbei ist es nicht relevant, aus welchem Simulationstool die FMU exportiert wurde. In dieser Arbeit wurden MATLAB Simulink und OpenModelica verwendet.

5.2.3 Schnittstelle zur Kommunikationssimulation

Zur Simulation der Kommunikation wurde OMNet++ verwendet. Hierbei erfolgt die Kommunikation zwischen Java und OMNet++ ebenfalls über eine Socket-Verbindung. OMNet++ erstellt einen Server, mit dem sich ein Client von Java aus verbindet. Da auf jede Anfrage des Clients von Java an den Server in OMNet++ eine Antwort erfolgt, nämlich an welche Komponentensimulation die Nachricht übermittelt wird, reichen ein Client und ein Server aus und es ist nicht notwendig einen Java-seitigen Server mit entsprechendem Client in OMNet++ aufzubauen. Um die Nachrichten in OMNet++ entgegen zu nehmen, wird in OMNet++ ein Interface-Knoten benötigt, der als Schnittstelle zum Server dient und die Nachrichten an die entsprechenden modellierten Knoten weiterleitet.

5.2.4 Schnittstelle zur Prozesssimulation

Zur Simulation der Umwelt und von physikalischen Prozessen wurde Unity 3D gewählt. Auch hier erfolgt die Kommunikation über Sockets. Bei der Prozesssimulation reicht ebenfalls ein Simulationstool-seitiger Server und ein Java-seitiger Client, da hier ebenfalls jede Anfrage des Clients eine Antwort erhält und die Simulation selbst nicht Kontakt aufnehmen muss. In Unity 3D wurde der Server in C# implementiert.

5.3 Implementierung der Synchronisation

Die Implementierung der Synchronisation teilt sich in die Implementierung des Taktgebers und die Implementierung der einzelnen Synchronisationsschnittstellen auf.

5.3.1 Implementierung des Taktgebers

Der Taktgeber wurde auch in Form eines Agenten umgesetzt. Er sendet an alle anderen Agenten zu Beginn eines Synchronisationszyklus die Nachricht den nächsten Zeitschritt zu starten. Hierbei übermittelt er die Länge dieses Zeitschritts. Anschließend wartet er, bis alle Agenten die Beendigung des Simulationsschritts gemeldet haben, bevor er den nächsten Synchronisationszyklus startet.

Die Länge des Zeitschritts wurde in dieser Arbeit fest mit 2 Sekunden vorgegeben, da bisher noch nicht die Ermittlung des kleinstmöglichen Zeitschritts implementiert ist, wie sie in 3.4.5 beschrieben ist.

5.3.2 Realisierung der Synchronisation in den einzelnen Simulationstools

Im Folgenden werden die Synchronisationsschnittstellen und die Funktionsweise der Synchronisation in den einzelnen Simulationstools kurz beschrieben.

MATLAB Simulink

In MATLAB Simulink kann eine Simulation für einen bestimmten Schritt ausgeführt werden. Es wird von der spezifischen Synchronisationsschnittstelle ein Simulationsschritt gestartet und die Meldung der Beendigung des Schritts von MATLAB Simulink entgegengenommen.

OpenModelica

OpenModelica bietet keine Möglichkeit, eine Simulation zur Laufzeit, getriggert durch ein externes Programm, zu pausieren. Daher wurde keine Synchronisationsschnittstelle implementiert und es kann nur für zeitunkritische Simulationen verwendet werden, wie beispielsweise einfache Sensoren, die sehr schnell eine Antwort liefern.

AnyLogic

Da, wie in 5.2.2.3 beschrieben, es in AnyLogic nicht möglich ist, gleichzeitig zu simulieren und Daten auszutauschen, wurden die Zustände „Simulieren“ und „Datenaustausch“ eingeführt, die im Wechsel eingenommen werden. Beim Übergang in den Zustand „Simulieren“ wird der Simulation mitgeteilt, wie lang simuliert werden soll. Hierfür sendet die spezifische Synchronisationsschnittstelle die Zeit des nächsten Simulationsschritts an AnyLogic und wartet, bis ein erneuter Verbindungsaufbau mit dem Server in AnyLogic möglich ist.

FMI

FMI bietet die Möglichkeit, Simulationen in einer FMU mit Hilfe der Funktion „doStep“ schrittweise auszuführen. Diese Schritte können zeitbasiert sein. Somit kann die spezifische Synchronisationsschnittstelle diese Funktion aufrufen und die Beendigung des Simulationsschritts abwarten.

OMNet++

Für die Simulation der Kommunikation ist keine Synchronisation notwendig, da nur Nachrichten weitergeleitet werden und somit keine Kausalitätsfehler auftreten können. Diese Weiterleitung ist im Vergleich zu den physikalischen Prozessen so schnell, dass eine Nachricht quasi sofort nachdem sie eine Simulation verlassen hat bei der Empfängersimulation ankommen kann. Daher wurde für OMNet++ keine spezifische Synchronisation implementiert.

Unity 3D

Physikalische Prozesse können langsam ablaufen, weshalb für die Prozesssimulation eine Synchronisation benötigt wird. Eine Simulation kann in Unity nicht schrittweise ausgeführt werden, sie kann jedoch pausiert und fortgesetzt werden, zudem kann die Simulationszeit abgerufen werden. Theoretisch könnte somit die spezifische Synchronisationsschnittstelle regelmäßig die Simulationszeit abrufen und zum entsprechenden Zeitpunkt die Simulation pausieren. Durch die Socket-Verbindung kommt es allerdings zu Verzögerungen, was diese Vorgehensweise ungenau werden lässt. Daher ist es sinnvoller direkt in Unity 3D diese Abfrage der Simulationszeit mit Pausieren und Fortsetzen zu implementieren. Dies kann in C# geschehen.

5.4 Implementierung der Interaktionssimulationen

5.4.1 Kommunikationssimulation in OMNet++

Für die Kommunikationssimulation wurde OMNet++ gewählt, da es Schnittstellen nach außen bietet und eine Vielzahl an Kommunikationstechnologien simulieren kann. Zusätzlich wurde ein WiFi-Netzwerk simuliert, da WiFi eine verbreitete Technologie ist. Wie bereits in 5.2.3 beschrieben, wird ein Interfaceknoten benötigt, der die Nachrichten über den Server in OMNet++ von der spezifischen Kommunikationsschnittstelle entgegen nimmt (rot umrandet in Abbildung 38). Dieser Interfaceknoten übermittelt die angenommene Nachricht dann an den Knoten, der den Absender repräsentiert. Dieser ist durch eine IP-Adresse identifizierbar (siehe Abbildung 38). Anschließend wird in OMNet++ die Nachrichtenübermittlung an den Empfängerknoten simuliert. Von diesem wird die Nachricht wieder an den Interfaceknoten weitergeleitet, welcher sie dann über den Server an die spezifische Kommunikationsschnittstelle zurückgibt. Das Modell für die Übertragung wird von OMNet++ bereitgestellt, zudem ist es möglich zur Laufzeit Knoten anzulegen, diesen IP-Adressen zuzuweisen und Knoten zu entfernen. Ein Modell einer Kommunikationssimulation mit fünf Knoten ist in Abbildung 38 dargestellt.

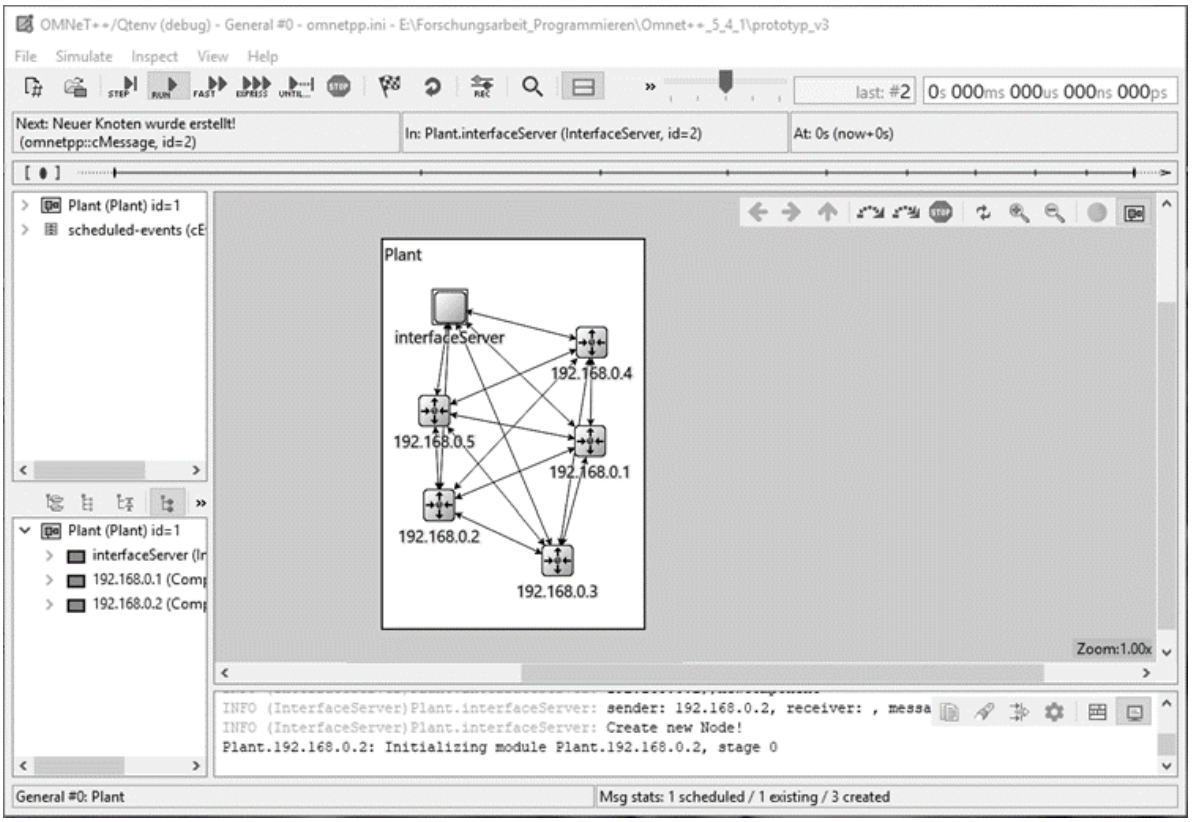


Abbildung 38: Kommunikationssimulation in OMNet++

5.4.2 Prozessinteraktion in Unity 3D

Zur Simulation der Umgebung und der physikalischen Prozesse wurde Unity 3D gewählt, da es bereits viele physikalische Prozesse vorimplementiert hat. Es konnten viele problemlos übernommen werden, wie beispielsweise Kräfte zwischen Komponenten, wie Reibung, oder Kräfte wie Trägheit und Schwerkraft (siehe Abbildung 39, Reibung zwischen Gabelstapler und Ware). Somit konnten Transporte von Komponenten problemlos modelliert werden. Einige physikalischen Prozesse, hauptsächlich indirekte Interaktionen wie Erwärmen oder Kühlen waren nicht vorimplementiert. Da Unity 3D aber sehr viele Freiheitsgrade bietet, war es möglich diese physikalischen Prozesse in C# zu implementieren (siehe Abbildung 39, Kühleinheit auf dem Regal und Temperatursensor unter dem Regal). Ein weiterer Vorteil von Unity 3D ist die grafische Darstellung von Objekten, durch die der Ablauf einer Co-Simulation und das Zusammenwirken der einzelnen Komponenten besser nachvollziehbar ist (siehe Abbildung 39). Es ist zudem möglich, zur Laufzeit Objekte hinzuzufügen und zu entfernen.

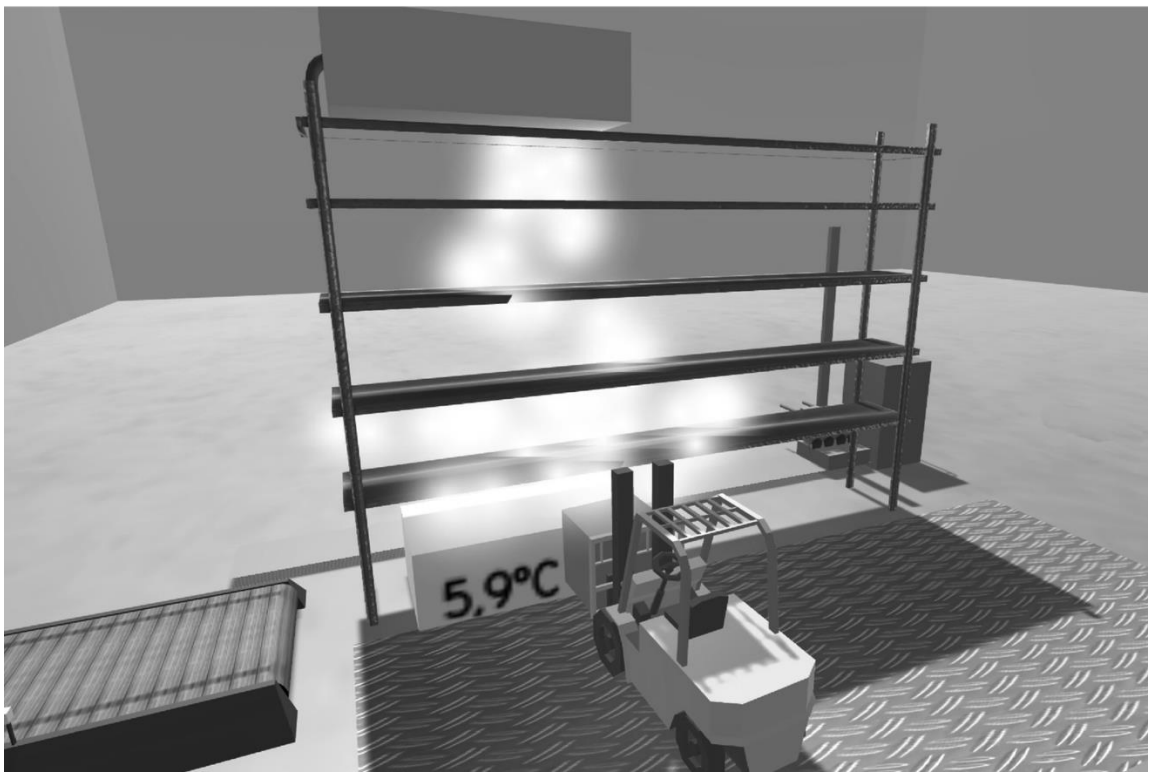


Abbildung 39: Prozesssimulation eines Warenlagers in Unity 3D

In diesem Kapitel wurde die Implementierung des in Kapitel 3 präsentierten Konzepts als Framework vorgestellt. Zuerst wurde der Kern des Frameworks, bestehend aus Co-Simulationsumgebung, Simulationsvertretern und Benutzeroberfläche präsentiert. Für diese Implementierung wurde Java bzw. Jadex gewählt. Anschließend wurde die Implementierung der Schnittstellen beschrieben gefolgt von der Implementierung der Synchronisation. Abschließend wurde die Implementierung der Interaktionssimulationen vorgestellt. Anhand dieser Implementierung wird im folgenden Kapitel das Konzept in Bezug auf die in 1.3 aufgestellten Anforderungen und Mehrwerte evaluiert.

Nach der Methode der Design Science wurden auch mehre Iterationen über die Implementierung durchgeführt. Hierbei wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie unterstützt. Die einzelnen Implementierungen der Schnittstellen zu den einzelnen Simulationstools geschah in mehreren Iterationen. Auch die Auswahl geeigneter Simulationstools geschah in mehreren Iterationen, unterstützt durch die Diskussion mit den Experten aus der Industrie. Auch die Benutzeroberfläche wurde in mehreren Iterationen erstellt, hierbei lag ein besonderes Augenmerk darauf, welche Informationen dargestellt werden sollten sowie auf einer einfachen Bedienbarkeit. Nach jeder Iteration wurde eine Kontrolle anhand der aufgestellten Anforderungen durchgeführt sowie auf die Tauglichkeit der Implementierung hinsichtlich eines zukünftigen Einsatzes in der Industrie über die genannten Diskussionen mit Experten aus der Industrie.

6 Evaluierung der Co-Simulation

In diesem Kapitel wird die Evaluierung des in Kapitel 3 vorgestellten Konzept beschrieben. Grundlage für die Evaluierung ist das in den Kapiteln 4 und 5 beschrieben Co-Simulationsframework für eine „Plug-and-Simulate“-fähige Co-Simulation. Zusätzlich zu dem Co-Simulationsframework werden für die Evaluierung noch ein Co-Simulationsszenario und damit die dazugehörigen Modelle und der Ablauf der Simulation benötigt.

Daher wird zunächst in 6.1 das Simulationsszenario beschrieben, welches zur Evaluierung verwendet wurde. Hierzu wird zunächst das simulierte IoT-System beschrieben, danach werden die verwendeten Modelle beschrieben und abschließend der Ablauf der Simulation. Anhand des Simulationsablaufs wird beschrieben, welche Aspekte welche Forschungsanforderungen abbilden.

Die anschließende Evaluierung unterteilt sich in zwei Teile. In 6.2 wird anhand des implementierten Frameworks nachgewiesen, dass das vorgestellte Konzept die in 1.3 aufgestellten Herausforderungen erfüllt. Dies geschieht anhand das in 6.1 beschriebenen Szenarios und dessen Ablauf sowie den für die Durchführung der Simulation benötigten Tätigkeiten.

In 6.3 wird nachgewiesen, dass das Konzept und das daraus resultierende Framework die Mehrwerte aus Kapitel 1.3 bietet. Hierzu dienen ebenfalls das in 6.1 beschriebene Szenario und dessen Ablauf sowie den für die Durchführung der Simulation benötigten Tätigkeiten. Zusätzlich werden hierfür die Ergebnisse der im Rahmen dieser Arbeit betreute studentische Arbeiten verwendet.

Nach der Methode der Design Science wurden auch mehre Iterationen über die Evaluierung durchgeführt. Hierbei wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie unterstützt. Hauptsächlich wurden geeignete Simulations- und Evaluierungsszenarien iteriert und welche Aspekte diese abdecken sollten. Nach jeder Iteration wurde eine Kontrolle anhand der aufgestellten Anforderungen durchgeführt sowie auf die Relevanz hinsichtlich eines zukünftigen Einsatzes in der Industrie über die genannten Diskussionen mit Experten aus der Industrie.

6.1 Beschreibung des Szenarios

In diesem Unterkapitel wird das zur Evaluierung verwendete Szenario beschrieben. Hierzu wird zuerst das simulierte IoT-System beschrieben und anschließend die für eine Co-Simulation des beschriebenen IoT-Systems benötigten Modelle. Abschließend wird der Ablauf der Simulation dargestellt.

6.1.1 Simuliertes IoT-Szenario

Das gewählte Szenario stellt ein Warenlager dar. Gründe für die Wahl dieses Szenarios waren:

- Es beinhaltet die elementaren Anwendungsfälle, die typisch für ein IoT-System sind: Objekt-Tracking (Tracking von Gabelstaplern und Waren) und Sensornetzwerke (Temperatursensor und Kontaktsensoren im Lagerregal), diese wurden in 2.2.2 genauer beschrieben.
- Mit diesem Szenario kann das Konzept auf die Erfüllung der in 1.3 gestellten Herausforderungen und Mehrwerte hin evaluiert werden, dies wird genauer in 6.1.4 beschrieben.
- Die enthaltenen Komponenten können einfach modelliert werden. Dadurch stehen eine Vielzahl an Simulationstools zur Modellierung zur Verfügung, da man nicht extrem spezialisierte Simulationstools benötigt.

Das Warenlager besteht aus folgenden Komponenten:

- Wareneingang: über diesen kommen einzulagernde Waren an, die sich in kühl zu lagernde und normal zu lagernde Waren unterteilen.
- Mehrere Gabelstapler: diese holen die Waren am Wareneingang ab und lagern sie in das Lagerregal ein.
- Lagerregal mit Lagerverwaltung: dieses weist den Waren einen Platz zu und erkennt welche Plätze momentan belegt sind.
- Kühleinheit: diese kühlt einen Teil des Lagerregals.
- Temperatursensor: dieser misst die Temperatur im gekühlten Teil des Lagerregals.
- Lagerbediengerät: dieses lagert Waren aus dem Lagerregal aus.

In Abbildung 40 ist ein Überblick über das Warenlager, wie es in der Prozesssimulation dargestellt ist, inklusive der vorgestellten Komponenten zu sehen.

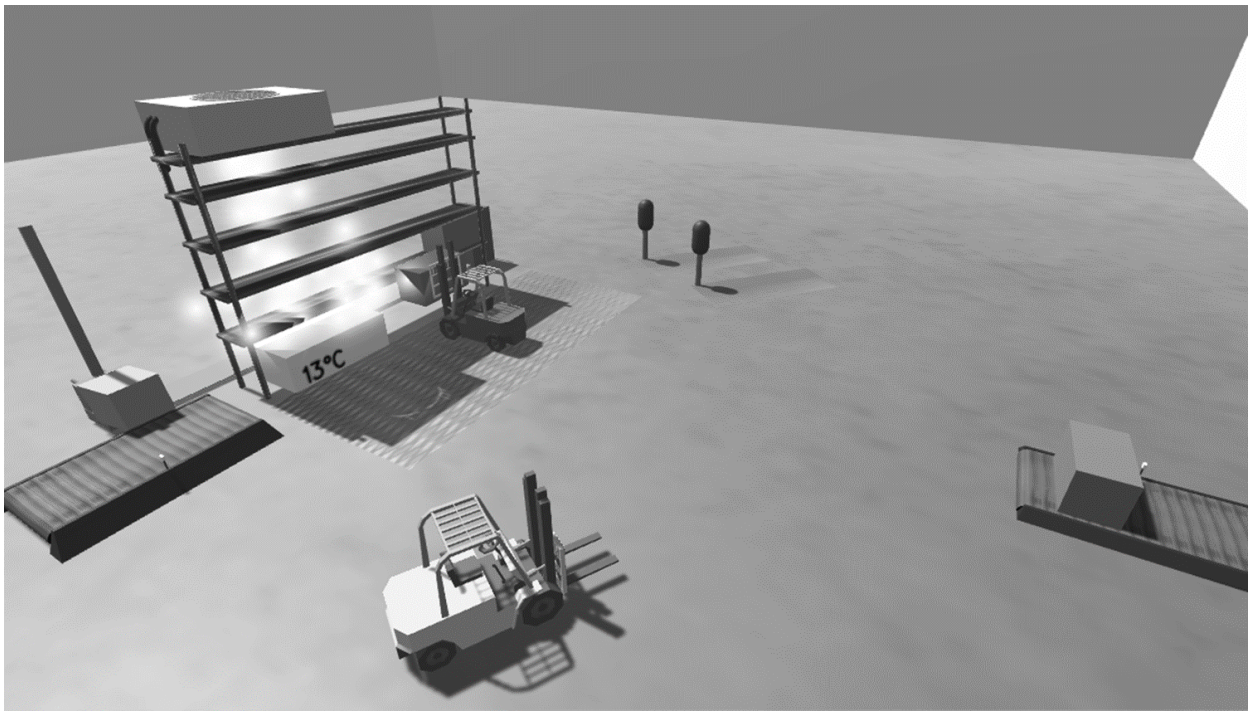


Abbildung 40: Überblick über das Evaluierungsszenario in der Prozesssimulation

6.1.2 Ziel des Simulationsexperiments und Vorgehen

Ziel der Simulation

Mit der Simulation soll erprobt werden, ob die vorhandenen Gabelstapler ausreichen, um die ankommenden Pakete einzulagern und ob das Lagerbediengerät schnell genug auslagern kann. Des Weiteren soll erprobt werden, ob ein Gabelstapler eines anderen Herstellers eine bessere Leistung erzielt. Hierzu werden zwei unterschiedliche Gabelstapler zeitgleich eingebunden.

Ziel des Simulationsexperiments

Anhand des Simulationsszenarios soll nachgewiesen werden, dass:

IoT-Komponenten getrennt voneinander in die Gesamtsimulation integrierbar sind:

Die Bestrebungen hin zur Rekonfigurierbarkeit bringen eine höhere Modularisierung mit sich, um in „Plug-and-Produce“-Szenarien möglichst leicht einzelne Komponenten austauschen zu können. Es ist wünschenswert, diese einfache Austauschbarkeit ebenfalls auf die Simulationswelt zu übertragen, so dass im Falle einer Rekonfiguration nicht nur die realen Komponenten per „Plug-and-Produce“ ausgetauscht werden können, sondern auch deren Modelle und somit deren Simulationen einfach per „Plug-and-Simulate“ ausgetauscht werden können. Somit ist es das Ziel, keine komplexe ineinander verwobene Einzelsimulation zu haben, sondern ebenfalls eine modular, komponentenorientiert aufgebaute Simulation.

Modelle und deren Simulationen unabhängig voneinander agieren:

Bestrebungen hin zu Autonomie sind ein weiterer Aspekt der Rekonfigurierbarkeit, mit dem Ziel, dass die einzelnen Komponenten eines Systems unabhängig voneinander autonom Aufgaben erfüllen können, ohne bei Eintritt einer derartigen Komponente manuelle Einstellungen und Anpassungen vornehmen zu müssen (Plug-and-Produce). Interaktionen zwischen Komponenten sollten ohne vorherigen menschlichen Eingriff erfolgen. Somit ist auch in der Simulation das Ziel

diese funktionale Unabhängigkeit zu erreichen, so dass bei Eintritt einer neuen Simulation diese nahtlos mit anderen Simulationen interagieren kann und somit manuelle Eingriffe unnötig sind.

Verschiedene Simulationstools aus unterschiedlichen Domänen verwendet werden:

Durch die immer höher werdende Vernetzung, auch über Domänengrenzen hinweg, werden IoT-Systeme immer heterogener. Diese Heterogenität wirkt sich auch auf die Simulationswelt aus: für die Simulation der einzelnen Komponenten werden verschiedene Tools aus unterschiedlichen Domänen verwendet.

Simulationen und deren Modelle zur Laufzeit eintreten:

Simulationen werden zunehmend während des gesamten Lebenszyklus eingesetzt und somit auch während des Betriebs eines Systems. Es ist daher von Vorteil, wenn, bedingt durch einen Eintritt einer neuen Komponente in das reale IoT-System zur Laufzeit, auch eine Simulation dieser neuen Komponente zur Laufzeit in die Simulation eintreten kann. Besonders während des Betriebs ist dies erstrebenswert, da hier bei IoT-Systemen, welche sehr dynamische Systeme sind, in die ständig neue Komponenten eintreten, beispielsweise in Form von neuen Produkten, oft die Zeit fehlt die gesamte Simulation zu stoppen und die neuen Teilsimulationen (auch Teilsimulationen, die zur Entwurfszeit der Simulation noch nicht bekannt waren) hinzuzufügen.

Vorgehen beim Simulationsexperiment

Das Vorgehen beim Simulationsexperiment unterteilt sich in zwei Aspekte, zum einen das Simulationsexperiment selber, in dem die Erfüllung der oben genannten Ziele experimentell nachgewiesen werden soll und zum anderen die Messung des Aufwandes zum Einbinden eines neuen Simulationstools.

Durchführung des Simulationsexperiments

Das Simulationsexperiment wurde mit dem im vorherigen Teilkapitel beschriebenen Szenario durchgeführt. Dabei wurden die vier beschriebenen Simulationsziele separate betrachtet.

Um nachzuweisen, dass IoT-Komponenten getrennt voneinander in die Gesamtsimulation integrierbar sind, wurden zur Laufzeit der Simulation unterschiedliche Simulationsteilnehmer eingefügt, diese waren:

- Gabelstapler mit unterschiedlichen Simulationstools modelliert
- Waren am Wareneingang
- Temperatursensor

Um nachzuweisen, dass Modelle und deren Simulationen unabhängig voneinander agieren wurden die Funktionen der einzelnen Modelle zuerst einzeln mit einem Testprogramm ausgeführt um nachzuweisen, dass sie selbst korrekt funktionieren. Anschließend wurden sie in die Gesamtsimulation eingefügt und deren dortiges Verhalten mit dem aus dem Testprogramm verglichen. So kann nachgewiesen werden, dass jedes Modell auch ohne andere Modelle funktionsfähig ist und unabhängig von anderen Modelle agieren kann.

Um nachzuweisen, dass verschiedene Simulationstools aus unterschiedlichen Domänen verwendet werden, wurden unterschiedliche Simulationstools eingesetzt und zur Laufzeit integriert. Zudem wurde dieselbe Komponente (Gabelstapler) in zwei unterschiedlichen Simulationstools (MATLAB Simulink und OpenModelica) modelliert und ausgeführt und das Verhalten der beiden Simulationen verglichen. Hierüber kann nachgewiesen werden, dass das selbe Verhalten in unterschiedlichen Simulationstools nachgebildet werden kann und somit auch unterschiedliche Simulationstools eingesetzt werden können.

Um nachzuweisen, dass Simulationen und deren Modelle zur Laufzeit eintreten wurden die folgenden Teilsimulationen zur Laufzeit eingefügt ohne die Gesamtsimulation oder die einzelnen Teilsimulationen zu stoppen:

- Gabelstapler mit unterschiedlichen Simulationstools modelliert
- Waren am Wareneingang
- Temperatursensor

Zusätzlich wurde in jedem Teil der Simulation das zeitliche Verhalten gemessen um nachzuweisen, dass eine Synchronität zwischen den einzelnen Teilsimulationen eingehalten wird. Hierzu wurden die einzelnen Simulationszeiten der einzelnen Teilsimulationen gemessen und aufgezeichnet.

Erfassung des Aufwands zum Einbinden eines neuen Simulationstools

Die Schnittstellen zum Einbinden eines neuen Simulationstools wurden durch studentische Arbeiten die im Rahmen dieser Dissertation betreut wurden durchgeführt. Die studentischen Arbeiten waren jeweils gleich aufgebaut um eine Vergleichbarkeit und Nachvollziehbarkeit zu gewährleisten. Zur Messung des Aufwands wurden für jede Implementierung der Aufwand zur Recherche des Simulationstools und dessen Schnittstellen erfasst, der Aufwand der Implementierung der Schnittstelle, der Aufwand zum Test dieser Schnittstelle sowie die der Aufwand bei der Dokumentation der Schnittstelle und anschließend addiert. Hierbei ist zu berücksichtigen, dass es sich um studentische Arbeiten handelt, und somit nicht um voll ausgebildete Software-Entwickler, somit ist anzunehmen, dass in einem industriellen Umfeld diese Aufwände wesentlich geringer sind, auch wenn dort deutlich intensiver getestet wird als bei einem Prototyp. Die Ergebnisse werden in Kapitel 6.3 in Tabelle 11 zusammengefasst.

6.1.3 Modellbeschreibung

Im Folgenden werden die einzelnen Modelle der Komponenten kurz beschrieben. Tabelle 10 gibt einen Überblick welche Komponente in welchem Simulationstool modelliert wurde.

Tabelle 10: Übersicht über die verwendeten Simulationstools

Komponente	Simulationstool
Wareneingang	AnyLogic
Gabelstapler 1	MATLAB Simulink
Gabelstapler 2	OpenModelica (angebunden über FMI)
Lagerverwaltung und Lagerregal	MATLAB Simulink
Kühleinheit	MATLAB Simulink (angebunden über FMI)
Temperatursensor	OpenModelica (angebunden über FMI)
Lagerbediengerät	MATLAB Simulink
Kommunikationssimulation	OMNet++
Prozesssimulation	Unity 3D

Wareneingang

Der Wareneingang wurde in AnyLogic modelliert, es werden zufällig Waren, die gekühlt, und Waren, die normal gelagert werden müssen, erzeugt. Sobald eine Ware bereitliegt, wird dies der Lagerverwaltung gemeldet.

Gabelstapler

Die Gabelstapler wurden in MATLAB Simulink und in OpenModelica (angebunden über FMI) modelliert, die unterschiedlichen Modellierungen stellen Gabelstapler von unterschiedlichen Herstellern da. Sie werden von der Lagerverwaltung beauftragt eine Ware abzuholen und einzulagern. Sie fahren dann bestimmte Checkpoints ab, dies geschieht, indem sie ihre Bewegungen an die Prozesssimulation melden.

Lagerverwaltung und Lagerregal

Die Lagerverwaltung beinhaltet das eigentliche Lagerregal und wurde in MATLAB Simulink modelliert. Sie koordiniert das Ein- und Auslagern und kommuniziert hierfür mit dem Wareneingang, den Gabelstaplern, dem Lagerbediengerät sowie dem Warenausgang. Das Lagerregal besitzt Sensoren, die erkennen, ob ein Platz belegt ist oder nicht.

Kühleinheit

Die Kühleinheit wurde in MATLAB Simulink (angebunden über FMI) modelliert. Sie erhält Temperaturwerte vom Temperatursensor und übermittelt entsprechend die Kühlleistung an die Prozesssimulation.

Temperatursensor

Der Temperatursensor wurde in OpenModelica (angebunden über FMI) modelliert. Er misst Temperaturwerte in der Prozesssimulation und meldet diese an die Kühleinheit.

Lagerbediengerät

Das Lagerbediengerät wurde in MATLAB Simulink modelliert und erhält von der Lagerverwaltung Warenauslagerungsaufträge. Die auszulagernden Waren werden von der Lagerverwaltung zufällig bestimmt. Zusätzlich wurde die Steuerung des Lagerbediengeräts als reale Komponente auf einem Raspberry Pi 3 Model B umgesetzt. Das Lagerbediengerät ist eine zeitunkritische Komponente, da es von der Lagerverwaltung nur den Auftrag bekommt ein Paket auszulagern und dies dann ausführt, während der Ausführung sind keine weiteren Interaktionen mit der Co-Simulation notwendig und somit kann die Steuerung in realer Zeit ablaufen.

Interaktionssimulationen

Die Interaktionssimulationen wurden in 5.4.1 (Kommunikationssimulation) und in 5.4.2 (Prozesssimulation) detailliert beschrieben.

Die einzelnen Komponentenmodelle und die Steuerung des Lagerbediengeräts sind im Anhang angefügt.

OpenModelica wurde in diesem Szenario nicht verwendet, da, wie in 5.3.2 erläutert, es keine einfache Synchronisation zulässt. Dieses Problem konnte über das Erstellen einer FMU umgangen werden, weshalb OpenModelica nur über FMI eingebunden wurde und nicht direkt.

6.1.4 Ablauf des Evaluierungsszenarios und der Simulation

Für den Ablauf des Evaluierungsszenarios wird die Co-Simulation gestartet, es werden die Interaktionssimulationen eingebunden und anschließend die Komponentensimulationen. Vorerst wird nur ein Gabelstapler eingebunden. Es werden zufällig Pakete erzeugt, die der Gabelstapler einlagert und das Lagerbediengerät wieder zufällig auslagert. Die Dauer des Einlagerungsprozesses ist aufgrund der Wege des Gabelstaplers höher als die gewünschte Zeit und es kommt zu Verzögerungen am Wareneingang. Daraufhin wird ein weiterer Gabelstapler eines anderen Herstellers, modelliert in einem anderen Simulationstool, zur Laufzeit hinzugefügt. Somit wurde eine Teilsimulation per „Plug-and-Simulate“ hinzugefügt. Es wird dann untersucht,

welcher Gabelstapler bessere Ergebnisse liefert, der schlechtere wird entfernt und durch einen Gabelstapler des bessern Typs ersetzt.

6.2 Erfüllung der Herausforderungen

Mit dem in 6.1 beschriebenen Szenario soll nachgewiesen werden, dass das in Kapitel 3 vorgestellte Konzept und die in 1.2 hergeleiteten Herausforderungen und die daraus in Kapitel 1.3 abgeleitete Zielsetzung erfüllt werden. Als Kriterien zur Überprüfung ob die Herausforderungen und somit auch die Zielsetzung erfüllt werden dienen die in Kapitel 1.3 hergeleiteten Anforderungen. Diese werden im Folgenden im Einzelnen diskutiert.

Im Folgenden werden die Herausforderungen im Einzelnen diskutiert.

Herausforderung 1 (H1): Die Simulationen der einzelnen IoT-Komponenten sollten getrennt voneinander in die Gesamtsimulation integrierbar sein.

Die aus dieser Herausforderung abgeleitete Anforderung A1 besagt, dass es möglich sein muss, Modelle und deren Simulationen hinzuzufügen oder zu entfernen, ohne dass das Gesamtmodell manuell angepasst werden muss. Dies wird im Evaluierungsszenario anhand der Gabelstapler gezeigt, indem deren Teilsimulationen und somit Teilsimulationen von IoT-Komponenten eingefügt werden, ohne das Modell des Gabelstaplers, das Modell der Kommunikationssimulation, das Modell der Umgebungssimulation oder Modelle anderer Komponenten zu verändern oder anzupassen. Genauso können, wie ebenfalls anhand der Gabelstapler gezeigt, Teilsimulationen zur Laufzeit entfernt werden, beides wird durch die Kapselung durch Simulationsvertreter ermöglicht.

Somit ist es möglich, die Simulationen der einzelnen IoT-Komponenten getrennt voneinander in die Gesamtsimulation zu integrieren und diese Herausforderung wird durch das in Kapitel 3 vorgestellte Konzept erfüllt.

Herausforderung 2 (H2): Modelle und deren Simulationen sollten unabhängig voneinander agieren können.

Die aus dieser Herausforderung abgeleitete Anforderung A2 besagt, dass es möglich sein muss, Modelle und deren Simulationen hinzuzufügen oder zu entfernen, ohne dass die möglichen Interaktionen mit anderen Modellen und deren Simulationen manuell angepasst werden müssen. Dies wird im Evaluierungsszenario anhand der Gabelstapler gezeigt, indem deren Teilsimulationen, und somit Simulationen von IoT-Komponenten, eingefügt werden, ohne die Interaktionen zwischen anderen Modellen zu verändern oder neue Interaktionen für die eingefügte Teilsimulation zu erstellen oder bestehende anzupassen. Hierbei müssen weder die Teilmodelle der anderen Komponenten noch die Modelle der Interaktionssimulationen angepasst werden. Genauso können, wie ebenfalls anhand der Gabelstapler gezeigt, Teilsimulationen zur Laufzeit

entfernt werden. Beides wird durch die getrennte Modellierung der Interaktionen in eigenen Interaktionssimulationen sowie durch die Kapselung durch Simulationsvertreter ermöglicht.

Somit können die Modelle und der Simulationen, bei entsprechender Modellierung der Teilsimulationen (welche allerdings ausschließlich vom Anwender abhängt), unabhängig voneinander agieren, da kein manueller Anpassungsaufwand an den Interaktionssimulationen und den Teilsimulationen notwendig ist und sie somit nicht abhängig von der Modellierung der Interaktionssimulationen und der anderen Teilsimulationen sind. Daher wird diese Herausforderung ebenfalls durch das Konzept erfüllt.

Herausforderung 3 (H3): Es sollen verschiedene Simulationstools aus unterschiedlichen Domänen verwendet werden können.

Die aus dieser Herausforderung abgeleitete Anforderung A3 besagt, dass es möglich sein muss die Teilmodelle in unterschiedlichen Simulationstools aus unterschiedlichen Domänen zu simulieren. Dies wird anhand des Evaluierungsszenarios und der prototypischen Umsetzung des Konzepts gezeigt, indem unterschiedliche Simulationstools aus unterschiedlichen Domänen eingebunden wurden. Zudem wurde eine Anbindung von Simulationstools über FMI demonstriert. Die verwendeten Simulationstools stammen aus folgenden Domänen:

- Logistik: AnyLogic
- Kommunikationsnetze: OMNet++
- Automobilbranche: FMI
- MATLAB Simulink, OpenModelica und Unity 3D sind nicht bestimmten Domänen zugeordnet.

Somit wurde nachgewiesen, dass sowohl unterschiedliche Simulationstools eingebunden werden können, als auch, dass diese aus unterschiedlichen Domänen stammen können. Daher wird diese Herausforderung durch das Konzept erfüllt.

Herausforderung 4 (H4): Simulationen und deren Modelle sollen zur Laufzeit eintreten können.

Die aus dieser Herausforderung abgeleitete Anforderung A3 besagt, dass es möglich sein muss die Gesamtsimulation zur Laufzeit um weitere Teilsimulationen zu erweitern. Dies wird im Evaluierungsszenario gezeigt, indem die Teilsimulationen der Gabelstapler zur Laufzeit eingefügt werden, ohne eine Teilsimulation oder die Gesamtsimulation zu stoppen oder zu pausieren. Dies wird durch die Kapselung durch die Simulationsvertreter ermöglicht. Das Einbinden von Teilsimulationen ist unabhängig vom gewählten Simulationstool, da dieses durch die Simulationsvertreter gekapselt wird. Zudem ist es möglich, auch zur Entwurfszeit der Co-

Simulation unbekannte Teilsimulationen einzubinden, dies wird durch die getrennte Modellierung der Interaktionen in eigenen Interaktionssimulationen ermöglicht.

Somit ist es möglich, dass Simulationen und deren Modelle zur Laufzeit in die Gesamtsimulation eintreten, ohne die Gesamtsimulation zu stoppen oder zu pausieren, weshalb auch diese Herausforderung durch das Konzept erfüllt wird.

Es wurde somit gezeigt, dass das in Kapitel 3 vorgestellte Konzept alle in 1.2 aufgestellten Herausforderungen erfüllt.

Im nächsten Kapitel wird, durch die Untersuchung hinsichtlich der in Kapitel 1.3 hergeleiteten Mehrwerte, gezeigt, dass auch die in Kapitel 1.3 hergeleitete Zielstellung dieser Arbeit erfüllt wird.

6.3 Erfüllung der Zielsetzung

Die Zielsetzung dieser Arbeit ist es, eine Antwort auf die Forschungsfrage zu geben:

Wie können Teilsimulationen mit minimalen manuellem Aufwand zur Laufzeit in eine Gesamtsimulation eines IoT-Systems integriert werden?

Außer der Erfüllung der Herausforderungen, wie in 6.2 gezeigt, muss hierfür nachgewiesen werden, dass die in 1.3 beschriebenen Mehrwerte erfüllt werden.

Hierfür wurde ein Vergleich hinsichtlich der Erfüllung dieser Mehrwerte mit FMI, stellvertretend für bestehende Co-Simulationsstandards, durchgeführt. Dieser Vergleich mit FMI erfolgt anhand verschiedener Kriterien, welche jeweils bei der Ausführung der einzelnen Mehrwerte im Folgenden erläutert werden. FMI wurde für diesen Vergleich gewählt, da es ein etablierter Co-Simulationsstandard ist, welcher im industriellen Umfeld zum Einsatz kommt.

Das Kriterium, über welches die Erfüllung des Mehrwerts M1 (Es soll möglich sein, mit minimalem manuellen Aufwand neue Teilsimulationen in eine Gesamtsimulation zur Laufzeit einzubinden) nachgewiesen wird ist die Erfüllung der Herausforderungen H1, H2 und H4. Herausforderungen H1 und H2 zusammengefasst fordern, dass kein manueller Aufwand bei der Einbindung einer neuen Teilsimulation, an den einzelnen Teilmodellen, der Gesamtsimulation oder an den Interaktionen der einzelnen Teilmodellen und somit auch an den Interaktionsmodellen erforderlich sein darf. Herausforderung H4 fordert, dass das Einbinden von Simulationen zur Laufzeit möglich sein soll, ohne die anderen Simulationen pausieren oder stoppen zu müssen. Somit ergibt sich aus der Kombination dieser drei Herausforderung der Mehrwert M1: Es soll möglich sein, mit minimalem manuellen Aufwand neue Teilsimulationen in eine Gesamtsimulation zur Laufzeit einzubinden.

Da, wie in 6.2 nachgewiesen, diese drei Herausforderungen von dem vorgestellten Konzept erfüllt werde, wird dieser Mehrwert ebenfalls erfüllt. Vergleicht man dies mit FMI, so zeigt sich, dass dieser Mehrwert von FMI nicht erfüllt wird. FMI muss beim Einbinden einer neuen Teilsimulation pausiert und anschließend neu gestartet werden, siehe [133]. Auch bei HLA müssen die Interaktionen neu angelegt werden, was kein nahtloses Einbinden von Teilsimulationen zur Laufzeit erlaubt. Dieser Vergleich zeigt den hauptsächlichen Vorteil und Mehrwert des vorgestellten Konzepts.

Das Kriterium, welches für den Nachweis der Erfüllung des zweiten Mehrwerts M2 (Es soll möglich sein, unterschiedliche Simulationstools verwenden zu können um die Teilsimulationen auszuführen) verwendet wird, ist die Anzahl der verwendbaren Simulationstools aus unterschiedlichen Domänen. Es ist möglich, wie in 6.2 nachgewiesen, unterschiedliche Simulationstools aus unterschiedlichen Domänen zu verwenden. Dies wurde anhand der Simulationstools MATLAB Simulink, OpenModelica, AnyLogic (Logistikbranche), OMNet++ (Kommunikationsnetze) und Unity 3D gezeigt. Zusätzlich wurde eine Anbindung von Simulationstools über FMI geschaffen (Automobilbranche).

In einem Vergleich mit FMI über die Anzahl der verwendbaren Simulationstools zeigt sich, dass das vorgestellt Konzept in diesem Punkt etablierten Co-Simulationsstandards nicht unterlegen ist. Laut offizieller Webseite von FMI sind über 150 Simulationstools an FMI angebunden. Durch die Anbindung von Simulationstools über eine FMU, ist es mit dem vorgestellten Konzept ebenfalls möglich alle in FMI enthaltenen Simulationstools anzubinden. Zusätzlich können noch weitere Simulationstools direkt angebunden werden, gezeigt mit MATLAB Simulink, OpenModelica, AnyLogic, OMNet++ und Unity 3D. Somit bietet das vorgestellte Konzept hinsichtlich dieses Vergleichs leichte Vorteile.

Das Kriterium, welches für den Nachweis der Erfüllung des dritten Mehrwerts M3 (Es soll möglich sein, neue Simulationstools mit einem möglichst geringen Aufwand an die Gesamtsimulation anzubinden) verwendet wird, ist der Zeitaufwand, gemessen in Mannwochen, für die Anbindung eines neuen Simulationstools.

Die gemessenen Zeitaufwände um die jeweiligen Schnittstellen zu implementieren für die ausgewählten Simulationstools sind in Tabelle 11 dargestellt. Die Schnittstellen wurden in sich abgeschlossenen Arbeiten im Rahmen dieser Arbeit von Personen durchgeführt, die zwar über Grundkenntnisse in der Softwareentwicklung verfügten, aber keine umfassende Erfahrung hatten.

Tabelle 11: Aufwände zur Einbindung der einzelnen Simulationstools

Simulationstool	Zeitaufwand
MATLAB Simulink	4 Personenwochen
OpenModelica	4-5 Personenwochen
AnyLogic	6-8 Personenwochen
OMNet++	5 Personenwochen
Unity 3D	1-2 Personenwochen
Anbindung an FMI	4 Personenwochen

Pro Schnittstelle entwickelte eine Person mit Grundkenntnissen aber ohne umfassende Erfahrung, eine Einarbeitung in die Tools war notwendig

Wie Tabelle 11 zu entnehmen ist, unterscheiden sich die Aufwände für die einzelnen Simulationstools nicht sonderlich, sondern alle lagen im Bereich von einigen wenigen Mannwochen. Da die implementierenden Personen zwar über Grundkenntnisse in der Softwareentwicklung verfügten, aber keine umfassende Erfahrung hatten, ist anzunehmen, dass in einem industriellen Umfeld diese Aufwände wesentlich geringer sind, auch wenn hier deutlich intensiver getestet wird als bei einem Prototyp. Das Vorgehen zur Erfassungen der zeitlichen Aufwände wird in Kapitel 6.1.2 beschrieben.

Somit bewegt sich der Aufwand zur Einbindung eines neuen Simulationstools in einem akzeptablen Rahmen, zumal dieser Aufwand pro Simulationstool nur einmal durchgeführt werden muss.

Bei FMI kann das Einbinden eines Simulationstools nur durch den Hersteller des Simulationstools erfolgen, bei dem vorgestellten Konzept hingegen kann es durch den Simulationstoolhersteller, den Co-Simulationsplattformbetreiber und durch den Co-Simulationsnutzer geschehen. Somit bietet das vorgestellt Konzept auch hinsichtlich dieses Vergleichs Vorteile.

In allen Vergleichen bietet das vorgestellte Konzept Vorteile, bzw. im Fall von Mehrwert M2 leichte Vorteile gegenüber etablierten Co-Simulationsstandards (hier stellvertretend gegenüber FMI). Tabelle 12 fasst die drei Vergleiche zusammen.

Tabelle 12: Vergleich des Dynamischen Co-Simulationskonzepts mit FMI

Mehrwert	**FMI, stellvertretend für etablierte Co-Simulationsstandards**	**Dynamisches Co-Simulationskonzept**
Aufwand zum Einbinden einer neuen Teilsimulation (M1)	- Co-Simulation muss gestoppt werden und Interaktionen müssen manuell zugewiesen werden	+ „Plug-and-Simulate“, vorausgesetzt: Interaktionen sind modelliert
Menge der verfügbaren Simulationstools (M2)	0 Über 150 integrierte Simulationstools	+ Anbindung von FMI: alle FMI-Tools plus weitere Schnittstellen
Aufwand zum Anbinden eines neuen Simulationstools (M3)	- Kann nur vom Hersteller des Simulationstools gemacht werden	+ Einmaliges Implementieren einer Schnittstelle pro Tool (ca. 4-5 Wochen)

- Schwäche des Ansatzes 0 Neutral + Stärke des Ansatzes

Das präsentierte Konzept zur dynamischen Co-Simulation erfüllt somit nicht nur die in 1.3 aufgestellten Herausforderungen, sondern verglichen mit einem etablierten Co-Simulationsstandard zeigen sich die Mehrwerte und Vorteile des präsentierten Konzepts. Es können, im Gegensatz zu etablierten Co-Simulationsstandards, Teilsimulationen zur Laufzeit in eine Co-Simulation eingebunden werden ohne manuellen Modellierungsaufwand an Teilmodellen oder an den Interaktionen zwischen diesen. Die Anzahl der verfügbaren Simulationstools entspricht der des etablierten Simulationsstandards und das Einbinden eines neuen Simulationstools gestaltet sich deutlich einfacher als bei gängigen Co-Simulationsstandards.

Das präsentierte Framework erleichtert den Entwicklungsaufwand für Ingenieure entscheidend, da durch das Konzept zur Erstellung der Schnittstellen, eine hohe Wiederverwendbarkeit für neue Simulationstools erzielt werden kann und somit mit jedem hinzugefügten Simulationstool der Aufwand für das Hinzufügen eines weiteren Simulationstools geringer wird.

Darüber hinaus fand die Erstellung des Konzepts sowie die Implementierung des Frameworks im Rahmen eines Industrieprojekts statt, wodurch ein Transfer des Konzepts in die Industrie ermöglicht wurde und ein Prototyp mit einem Technology Readiness Level [182] von 4 aufgebaut werden konnte. Dementsprechend wird die Zielsetzung der Arbeit erfüllt, eine zur Laufzeit erweiterbare Co-Simulation zu ermöglichen.

7 Schlussbetrachtungen

7.1 Zusammenfassung der Ergebnisse und Bewertung

Moderne automatisierte Systeme werden, bedingt durch unterschiedlichste Faktoren, zunehmend komplexer: So vernetzen sich Systeme durch das Internet der Dinge über Domänengrenzen hinweg, zudem werden sie deutlich modularer und flexibler gehalten und können sich zur Laufzeit verändern um sich an neue Gegebenheiten anzupassen und durch Ansätze wie „Plug-and-Play“ können neue, zur Entwurfszeit unbekannte Komponenten zur Laufzeit in Systeme eintreten.

Durch diese erhöhte Komplexität wächst auch der Bedarf an Unterstützung durch Simulation während des gesamten Lebenszyklus eines Systems. Allerdings überträgt sich diese Komplexität auch auf die Simulation solcher Systeme. Die Vernetzung über Domänengrenzen hinweg erhöht die Heterogenität der Systeme und ebenso die Heterogenität in der Simulation, weswegen eine Simulation solcher Systeme in einem einzelnen Simulationstool nahezu unmöglich wird. Zudem können verschiedene Komponenten effizienter in spezialisierten Simulationstools modelliert werden und Experten verwenden die von ihnen präferierten Simulationstools. Daher wird für die Simulation komplexer, vernetzter Systeme eine Co-Simulation benötigt. Darüber hinaus muss in der Simulation auch die Dynamik, im Speziellen das Ein- und Austreten von Komponenten zur Laufzeit, berücksichtigt werden. Daher sollte die Co-Simulation ebenfalls zur Laufzeit möglichst einfach erweiterbar sein um Ansätze wie „Plug-and-Play“ abbilden zu können.

Es gibt zwar erste Ansätze zur Modellierung und Simulation von IoT-Systemen, allerdings betrachten diese meist nur einen bestimmten Aspekt der IoT-Systeme und können, da diese nur ein Simulationstool verwenden, nicht der Heterogenität gerecht werden. Bestehende Co-Simulationsstandards wie FMI und HLA können der Heterogenität zwar besser begegnen, allerdings werden sie nicht der Dynamik gerecht, da sie kein Einbinden von neuen Teilsimulationen zur Laufzeit per „Plug-and-Simulate“ erlauben. Co-Simulationsansätze in der Literatur können mit der geforderten Dynamik zwar deutlich besser umgehen, allerdings müssen beim Einbinden von neuen Teilsimulationen und vor allem bei zur Entwurfszeit der Co-Simulation unbekannten Teilsimulationen die Interaktionen zwischen den Teilsimulationen manuell angelegt werden, weshalb auch sie kein „Plug-and-Simulate“ zulassen.

Deshalb wurde in dieser Arbeit ein Konzept für eine „Plug-and-Simulate“-fähige Co-Simulation vorgestellt. Die einzelnen Teilsimulationen werden an Simulationsvertreter angebunden und durch sie gekapselt. Durch sie können sie zur Laufzeit in die Co-Simulationsumgebung ein- und austreten. Die Interaktionen werden in sogenannten Interaktionssimulationen, welche ebenfalls durch Simulationsvertreter gekapselt sind, entkoppelt. Hierdurch müssen keine neuen Interaktionen bei einem Eintritt einer neuen Teilsimulation angelegt werden. Die Interaktionssimulationen teilen sich auf in Kommunikationssimulation und Prozesssimulation,

um die beiden Interaktionsarten, Kommunikation und physikalische Prozesse, getrennt simulieren zu können. Die Synchronisation der Co-Simulation erfolgt durch eine konservative Synchronisation, gesteuert durch einen Taktgeber.

Realisiert wurde das Konzept durch ein Co-Simulationsframework. In diesem werden die Simulationsvertreter durch Softwareagenten realisiert, die die Möglichkeit haben, zur Laufzeit in ein Agentensystem ein- bzw. auszutreten. Des Weiteren bietet dieses Framework ein Konzept zur Umsetzung der Schnittstellen an die verschiedenen Simulationstools. Dieses Framework bietet einerseits eine Kategorisierung von Schnittstellentypen und eine Schablone, wie diese realisiert werden können, andererseits bietet es ein Konzept zur Befähigung weiterer Simulationstools, welche die Kommunikation nicht standardgemäß simulieren können. Zudem wird eine Anbindung an FMI vorgestellt. Der dritte Teil des Frameworks ist die Vorstellung einer Modellbibliothek, welche Interaktionsmodelle enthält.

Die Implementierung dieses Frameworks wurde durch Java bzw. Jadex für das Agentensystem umgesetzt. Diese Implementierung beinhaltet eine Anbindung an die Simulationstools MATLAB Simulink, OpenModelica, AnyLogic sowie eine Anbindung an FMI zur Simulation der Komponenten. Die Kommunikationssimulation wird in OMNet++ simuliert und die Prozesssimulation in Unity 3D.

Zur Evaluation des vorgestellten Konzepts für eine „Plug-and-Simulate“-fähigen Co-Simulation wurde ein Warenlager, bestehend aus Wareneingang, Gabelstaplern, Lagerverwaltung mit Lagerregal, Lagerbediengerät, Kühleinheit und Temperatursensor simuliert. Anhand dieses Szenarios konnte mit dem implementierten Framework gezeigt werden, dass ein Eintritt von neuen, zur Entwurfszeit unbekannten Teilsimulationen zur Laufzeit möglich ist, ohne Modelle oder Interaktionen zwischen Modellen anpassen zu müssen. Dies geht mit etablierten Co-Simulationsstandards nicht. Weiterhin konnte durch die Anbindung von FMI gezeigt werden, dass eine mindestens genauso große Anzahl an Simulationstools zur Verfügung steht wie bei etablierten Co-Simulationsstandards. Weiterhin konnte gezeigt werden, dass der Aufwand zur Anbindung eines neuen Simulationstools im akzeptablen Bereich liegt und, im Gegensatz zu Co-Simulationsstandards, sowohl von den Simulationstoolanbietern, den Co-Simulationsanbietern und den Co-Simulationsnutzer durchgeführt werden kann. Somit wird die Zielsetzung einer „Plug-and-Simulate“-fähigen Co-Simulation von dem präsentierten Konzept erfüllt.

Nach der Methode der Design Science wurden mehre Iterationen über die einzelnen Teile der präsentierten Arbeit durchgeführt. Zum einen wurden das Konzept, die Realisierungsmöglichkeiten, die Implementierung sowie die Evaluierung in sich iteriert, zum anderen wurde auch über alle genannten Phasen als gesamtes iteriert. Hierbei wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie unterstützt. Gerade die Verfeinerungen der einzelnen Teilkonzepte, deren Realisierung, Implementierung und Darstellung in einem geeigneten Evaluierungsszenario wurden mehrmals iteriert und

beeinflussten sich oftmals gegenseitig. Beispielsweise die Unterteilung in kommunikations- und prozessorientierte Interaktionen startete eine neue Iteration. Auch die Einführung der Synchronisation war der Beginn einer weiteren Iteration. Unterstütz wurden diese Iterationen durch Diskussionen mit Experten aus der Industrie. Zudem wurde am Ende jeder Iteration eine Kontrolle anhand der aufgestellten Anforderungen durchgeführt.

7.2 Ausblick

Der Fokus dieser Arbeit lag auf der prinzipiellen Ermöglichung einer „Plug-and-Simulate“-fähigen Co-Simulation. Aspekte wie Performanz und Skalierbarkeit wurden bisher außer Acht gelassen und bieten Potential für weiterführende Arbeiten.

Die Performanz der gesamten Co-Simulation kann durch einen ausgefeilteren Synchronisationsalgorithmus erhöht werden. Es wäre beispielsweise möglich optimistische Synchronisationsalgorithmen zu erproben, vor allem hinsichtlich ihrer Anwendbarkeit auf eine breite Masse an Simulationstools. Hierbei stellt sich hauptsächlich die Frage, erlauben die meisten Simulationstools eine optimistische Synchronisation und falls nicht, wie können diese Simulationstools dann befähigt werden, eine optimistische Synchronisation zu ermöglichen.

Um die Skalierbarkeit zu erhöhen, kann untersucht werden, wie sich das Co-Simulationsframework auf mehrere Rechner verteilen lässt. Agenten bieten prinzipiell diese Möglichkeit, allerdings wäre zu untersuchen, ob dies die Performanz beeinträchtigt. Eine Verteilung über mehrere Rechner ist deshalb erstrebenswert, da die meisten Simulationstools hohe Ansprüche an die Rechenleistung haben und herkömmliche PCs schon bei einer überschaubaren Anzahl an Simulationen an ihre Grenzen stoßen. Auch die Synchronisation spielt für die Skalierbarkeit eine Rolle, da diese eventuell begrenzt ist, wenn zu kleine Synchronisationsschritte gewählt werden, da kleine Synchronisationsschritte durch das erhöhte Datenaufkommen ebenfalls mehr Ressourcen benötigen. Zudem ist momentan der Datenaustausch zwischen den Teilsimulationen nicht limitiert. Hier wäre ebenfalls zu untersuchen, ob sich ein uneingeschränkter Datenaustausch negativ auf die Skalierbarkeit auswirken kann.

Zusätzlich kann die Echtzeitfähigkeit des Co-Simulationskonzepts verbessert werden um HiL-Simulationen durchführen zu können. Dies beschränkt sich allerdings nicht nur auf die Co-Simulationsumgebung, die Simulationsvertreter und die Schnittstellen, sondern auch die verwendeten Simulationstools müssen echtzeitfähig sein.

Weiterhin besteht die Möglichkeit das Konzept der Schnittstellen zur erweitern und verfeinern. Einerseits können weitere Schnittstellentypen neben Socket-Kommunikation und APIs untersucht werden. Andererseits können die wiederverwendbaren Bausteine für diese Schnittstellentypen präziser definiert werden und teilweise vorimplementiert werden, sodass sich der Aufwand zum Einbinden neuer Simulationstools verringert.

Literaturverzeichnis

[1] E. Westkämper and C. Löffler, “Strategien der Produktion,” *Strategien der Produktion*, 2016, doi: 10.1007/978-3-662-48914-7.

[2] R. Birkhofer *et al.*, *Life-Cycle-Management für Produkte und Systeme der Automation*. Frankfurt, M.: Zentralverb. Elektrotechnik- und Elektronikindustrie, Fachverb. Automation, 2010.

[3] C. Legat, D. Schütz, S. Feldmann, S. Lamparter, C. Seitz, and B. Vogel-Heuser, “Wandlungsfähige automation auf knopfdruck,” *Gaswaerme International*, vol. 63, no. 2, pp. 63–72, 2014, doi: 10.17560/atp.v55i05.250.

[4] H. Meier, S. Schröder, J. Velkova, and A. Schneider, “Fabrikplanung, produktionsmanagement, umformtechnik: Modularisierung als Gestaltungswerkzeug für wandlungsfähige produktionssysteme,” *WT Werkstattstechnik*, vol. 102, no. 4, pp. 181–185, 2012.

[5] M. Obst *et al.*, “Automatisierung im Life Cycle modularer Anlagen,” *atp edition - Automatisierungstechnische Praxis*, vol. 55, no. 01–02, p. 24, 2013, doi: 10.17560/atp.v55i01-02.234.

[6] M. Weyrich, A. Zeller, J.-P. Schmidt, A. Faul, and P. Marks, “Engineering und Betrieb Smarter Komponenten in IoT-Netzwerken für die Automatisierung der Produktion,” *VDE-Kongress 2016 -- Internet der Dinge*, 2016.

[7] J. Jasperneite, S. Hinrichsen, and O. Niggemann, “„Plug-and-Produce“ für Fertigungssysteme: Anwendungsfälle und Lösungsansätze,” *Informatik-Spektrum*, vol. 38, no. 3, pp. 183–190, 2015, doi: 10.1007/s00287-015-0877-x.

[8] N. Jazdi, *Cyber physical systems in the context of Industry 4.0*. Piscataway, NJ: IEEE, 2014. doi: 10.1109/AQTR.2014.6857843.

[9] E. A. Lee, “Cyber physical systems: Design challenges,” in *Proceedings - 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing, ISORC 2008*, 2008, pp. 363–369. doi: 10.1109/ISORC.2008.25.

[10] R. Baheti and H. Gill, “Cyber-physical Systems,” *The Impact of Control Technology*, 2011.

[11] L. Monostori, “Cyber-physical Production Systems,” *Procedia CIRP*, vol. 17, pp. 9–13, 2014, doi: 10.1016/j.procir.2014.03.115 T4 - Roots, Expectations and R&D Challenges M4 - Citavi.

[12] “Plattform Industrie 4.0 U6 - http://www.plattform-i40.de/I40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html.”

[13] I. Lee and O. Sokolsky, *Medical cyber physical systems*. Piscataway: I E E E, 2010. doi: 10.1145/1837274.1837463.

[14] S. Karnouskos, “Cyber-physical systems in the SmartGrid,” in *IEEE International Conference on Industrial Informatics (INDIN)*, Piscataway, NJ: IEEE, Jul. 2011, pp. 20–23. doi: 10.1109/INDIN.2011.6034829.

[15] J. Shi, J. Wan, H. Yan, and H. Suo, *A survey of Cyber-Physical Systems*. Piscataway, NJ: IEEE, 2011. doi: 10.1109/WCSP.2011.6096958.

[16] F. Xia, L. T. Yang, L. Wang, and A. Vinel, “Internet of things,” *International Journal of Communication Systems*, vol. 25, no. 9. Wiley Subscription Services, Inc., A Wiley Company, pp. 1101–1102, Sep. 01, 2012. doi: 10.1002/dac.2417.

[17] “http://www.beechamresearch.com/article.aspx?id=4 U6 - http://www.beechamresearch.com.”

[18] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges,” *Ad Hoc Networks*, vol. 10, no. 7. Elsevier B.V., pp. 1497–1516, Sep. 01, 2012. doi: 10.1016/j.adhoc.2012.02.016.

[19] L. Da Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Trans Industr Inform*, vol. 10, no. 4, pp. 2233–2243, 2014, doi: 10.1109/TII.2014.2300753.
[20] M. Traub, H.-J. Vögel, E. Sax, and J. Härri, *Digitalization in automotive and industrial systems; Digitalization in automotive and industrial systems*. 2018. doi: 10.23919/DATE.2018.8342198.
[21] J. Košturiak and M. Gregor, “Simulation in production system life cycle,” *Comput Ind*, vol. 38, no. 2, pp. 159–172, 1999, doi: 10.1016/S0166-3615(98)00116-X.
[22] J. Heilala *et al.*, *Simulation-based sustainable manufacturing system design*. Winter Simulation Conference, 2008. doi: 10.1109/WSC.2008.4736284.
[23] J. A. Ledin, “Hardware-in-the-Loop Simulation,” *Embedded Systems Programming*, no. 02, 1999.
[24] P. Hoffmann, R. Schumann, T. M. A. Maksoud, and G. C. Premier, *Virtual commissioning of manufacturing systems a review and new approaches for simplification*. [S.l.]: ECMS, 2010.
[25] R. K. Cox, J. F. Smith, and Y. Dimitratos, “Can simulation technology enable a paradigm shift in process control?. Modeling for the rest of us,” *Comput Chem Eng*, vol. 30, no. 10–12, pp. 1542–1552, 2006, doi: 10.1016/j.compchemeng.2006.05.020.
[26] M. Becker, S. Balci, and H. Szczerbicka, *Predictive simulation based decision support system for resource failure management in multi-site production environments*. Piscataway, NJ: IEEE, 2014. doi: 10.1109/CoDIT.2014.6996949.
[27] B. Ashtari Talkhestani *et al.*, “An architecture of an Intelligent Digital Twin in a Cyber-Physical Production System,” *At-Automatisierungstechnik*, vol. 67, no. 9, pp. 762–782, 2019, doi: 10.1515/auto-2019-0039.
[28] J.-P. Schmidt, A. Zeller, and M. Weyrich, “Ansatz zur modellgetriebenen Entwicklung flexibler Automatisierungssysteme durch Serviceorientierung,” *Eka 2016*, no. May, 2016.
[29] B. Heinzl, W. Kastner, I. Leobner, F. Dür, F. Bleicher, and I. Kovacic, *Using coupled simulation for planning of energy efficient production facilities*. Piscataway, NJ: IEEE, 2014. doi: 10.1109/MSCPES.2014.6842397.
[30] S. Grabmaier, M. Jüttner, D. Vögeli, W. M. Rucker, and P. Göhner, “Numerical framework for the simulation of dielectric heating using finite and boundary element method,” *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields*, vol. 31, no. 2, pp. 1–10, 2018, doi: 10.1002/jnm.2273.
[31] M. Oppelt, M. Barth, and L. Urbas, “The Role of Simulation within the Life-Cycle of a Process Plant - Results of a global online survey,” 2015. doi: 10.13140/2.1.2620.7523.
[32] M. Oppelt, L. Bruckner, R. Rosen, M. Barth, L. Urbas, and J. Jäkel, “Die aktuelle und zukünftige Nutzung von Simulation,” *atp magazin*, vol. 63, no. 04, pp. 64–73, Apr. 2021, doi: 10.17560/ATP.V63I04.2533.
[33] J. Um, S. Weyer, and F. Quint, “Plug-and-Simulate within Modular Assembly Line enabled by Digital Twins and the use of AutomationML,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 15904–15909, Jul. 2017, doi: 10.1016/J.IFACOL.2017.08.2360.
[34] D. Terrier, “2020 NASA Technology Taxonomy,” 2020.
[35] E. Commission, “FACTORIES OF THE FUTURE Policy Research,” 2020.
[36] VDI, “VDI-Richtlinie 3633,” 2013.
[37] M. Oppelt, G. Wolf, and L. Urbas, “Simulation im Lebenszyklus einer Prozessanlage,” *atp edition - Automatisierungstechnische Praxis*, vol. 57, no. 10, p. 38, 2015, doi: 10.17560/atp.v57i10.534.
[38] E. R. Ziegel and J. Banks, *Handbook of Simulation*, vol. 42, no. 2. New York, NY: Wiley, 2000. doi: 10.2307/1271478.
[39] A. Gupta, *Simulation modeling and analysis*, 4. ed., in. in McGraw-Hill series in industrial engineering and management science. Boston [u.a.]: McGraw-Hill, 2016. doi: 10.1201/9781315183176.

[40] R. McHaney, *Computer Simulation: A Practical Perspective*. Academic Press, 1991.
[41] R. Rosen, G. Von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *IFAC-PapersOnLine*, vol. 28, no. 3, pp. 567–572, 2015, doi: 10.1016/j.ifacol.2015.06.141.
[42] M. Shafto *et al.*, "DRAFT Modeling, Simulation, information Technology & Processing Roadmap - Technology Area 11," *National Aeronautics and Space Administration*, p. 27, 2010.
[43] M. Oppelt, G. Wolf, and L. Urbas, *Towards an integrated use of simulation within the life-cycle of a process plant: A prototypical implementation*, 1. Aufl., vol. 2015-Octob. in Berichte aus der Automatisierungstechnik, vol. 2015- Octob. Aachen: Shaker, 2015. doi: 10.1109/ETFA.2015.7301521.
[44] S. Boschert and R. Rosen, "Digital twin-the simulation aspect," in *Mechatronic Futures: Challenges and Solutions for Mechatronic Systems and Their Designers*, P. Hehenberger and D. Bradley, Eds., Cham: Springer International Publishing, 2016, pp. 59–74. doi: 10.1007/978-3-319-32156-1_6.
[45] D. M. Penfold, J. M. Hammersley, and D. C. Handscomb, *Monte Carlo Methods*, vol. 51, no. 378. in Monographs on Applied Probability and Statistics, vol. 51. Dordrecht: Springer Netherlands, 1967. doi: 10.2307/3612994.
[46] E. Yamaguchi, *Finite element method*. in ISTE. London: Wiley, 2014. doi: 10.1201/b15616.
[47] J. Misra, "Distributed Discrete-Event Simulation," *ACM Computing Surveys (CSUR)*, vol. 18, no. 1, pp. 39–65, Mar. 1986, doi: 10.1145/6462.6485.
[48] S. Raczynski, *Modeling and Simulation: The Computer Science of Illusion*. John Wiley & Sons, 2006.
[49] J. Busch, *Continuous Simulation with Ordinary Differential Equations*, no. 6140789. 2012.
[50] K. Binder and D. W. Heermann, *Monte Carlo Simulation Statistical An Introduction,Fifth Edition*, vol. 0, no. January. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-03163-2.
[51] T. J. Misa and T. J. Misa, *Industrial Dynamics*. Mansfield Centre, CT and Cambridge, MA: Martino and {MIT Press}, 2015. doi: 10.5749/minnesota/9780816683314.003.0008.
[52] H.-Joachim. Bungartz, Stefan. Zimmer, Martin. Buchholz, and Dirk. Pflüger, *Modellbildung und Simulation Eine anwendungsorientierte Einführung*. Imprint: Springer Spektrum, 2013.
[53] M. Weiser, "The Computer for the 21st Century," *Sci Am*, vol. 265, no. 3, pp. 94–104, 1991, doi: 10.1038/scientificamerican0991-94.
[54] K. Ashton, "That 'Internet of Things' Thing," *RFID Journal*, 2009.
[55] S. Deoras, "First Ever IoT Device- 'The Internet Toaster,'" *Iotindiamag*. 2016.
[56] F. Mattern and C. Flörkemeier, "Vom Internet der Computer zum Internet der Dinge," *Informatik-Spektrum*, vol. 33, no. 2, pp. 107–121, 2010, doi: 10.1007/s00287-010-0417-7.
[57] M. A. Salman, "On Identification of Internet of Things," *International Journal of Sciences: Basic and Applied Research (IJSBAR)*, vol. 18, no. 1, pp. 59–62, 2014.
[58] Gartner, "Gartner Says 6.4 Billion Connected 'Things' Will Be in Use in 2016, Up 30 Percent From 2015," *Gartner, Inc.* www.gartner.com, p. 1, 2015.
[59] H. Hada and J. Mitsugi, "EPC based internet of things architecture," in *2011 IEEE International Conference on RFID-Technologies and Applications, RFID-TA 2011*, [Piscataway, NJ]: IEEE, 2011, pp. 527–532. doi: 10.1109/RFID-TA.2011.6068595.
[60] N. Koshizuka, "Ubiquitous ID Technology," *IEEE Pervasive Comput*, vol. 9, no. 4, pp. 98–101, 2009, doi: 10.1109/MPRV.2010.87 T4 - Standards for Ubiquitous Computing and the Internet of Things M4 - Citavi.

[61] D. Giusto, A. Iera, G. Morabito, and L. Atzori, *The Internet of Things: 20th Tyrrhenian Workshop on Digital Communications*. 2010. doi: 10.1007/978-1-4419-1674-7.
[62] R. van Kranenburg and Sean. Dodson, *The internet of things : a critique of ambient technology and the all-seeing network of RFID*. Institute of Network Cultures, 2008.
[63] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010, doi: 10.1016/j.comnet.2010.05.010.
[64] M. Zareei, A. Zarei, R. Budiarto, and M. A. Omar, "A comparative study of short range wireless sensor network on high density networks," in *17th Asia-Pacific Conference on Communications, APCC 2011*, Piscataway, NJ: IEEE, Oct. 2011, pp. 247–252. doi: 10.1109/APCC.2011.6152813.
[65] C. Gomez, J. Oller, and J. Paradells, "Overview and evaluation of bluetooth low energy: An emerging low-power wireless technology," *Sensors (Switzerland)*, vol. 12, no. 9, pp. 11734–11753, 2012, doi: 10.3390/s120911734.
[66] S. Sudevalayam and P. Kulkarni, "Energy harvesting sensor nodes: Survey and implications," *IEEE Communications Surveys and Tutorials*, vol. 13, no. 3, pp. 443–461, 2011, doi: 10.1109/SURV.2011.060710.00094.
[67] W. D. Leon-Salas and C. Halmen, "A RFID Sensor for Corrosion Monitoring in Concrete," *IEEE Sens J*, vol. 16, no. 1, pp. 32–42, 2016, doi: 10.1109/JSEN.2015.2476997.
[68] S. Stocklin, T. Volk, A. Yousaf, J. Albesa, and L. Reindl, *Efficient inductive powering of brain implanted sensors*. Piscataway, NJ: IEEE, 2015. doi: 10.1109/SAS.2015.7133583.
[69] U. Isikdag, "Internet of things: Single-board computers," in *SpringerBriefs in Computer Science*, no. 9783319218243, Springer, 2015, pp. 43–53. doi: 10.1007/978-3-319-21825-0_4.
[70] J. A. Stankovic, "Research directions for the internet of things," *IEEE Internet Things J*, vol. 1, no. 1, pp. 3–9, 2014, doi: 10.1109/JIOT.2014.2312291.
[71] N. Brenner, A. Lauber, C. Meier, W. Reitmeier, and E. Sax, "Requirements of Automated Vehicles and Depots for the Initial Step of Automated Public Transport," pp. 15–26, 2021, doi: 10.1007/978-3-658-29717-6_2.
[72] D. M. Segura Velandia, N. Kaur, W. G. Whittow, P. P. Conway, and A. A. West, "Towards industrial internet of things," *Robot Comput Integr Manuf*, vol. 41, pp. 66–77, 2016, doi: 10.1016/j.rcim.2016.02.004.
[73] H. Schmitzberger and W. Narzt, "Leveraging WLAN infrastructure for large-scale indoor tracking," in *Proceedings - 6th International Conference on Wireless and Mobile Communications, ICWMC 2010*, 2010, pp. 250–255. doi: 10.1109/ICWMC.2010.57.
[74] S. S. Chawathe, "Beacon placement for indoor localization using Bluetooth," in *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 2008, pp. 980–985. doi: 10.1109/ITSC.2008.4732690.
[75] F. Yanhui, "Internet of things application of RFID technology in warehouse management," in *Proceedings - 3rd International Conference on Instrumentation and Measurement, Computer, Communication and Control, IMCCC 2013*, 2013, pp. 1697–1701. doi: 10.1109/IMCCC.2013.375.
[76] M. Boehme, M. Stang, F. Muetsch, and E. Sax, "TalkyCars: A Distributed Software Platform for Cooperative Perception; TalkyCars: A Distributed Software Platform for Cooperative Perception," 2020, doi: 10.1109/IV47402.2020.9304630.
[77] M. Weyrich, *Industrielle Automatisierungs- und Informationstechnik*. Springer Berlin Heidelberg, 2023. doi: 10.1007/978-3-662-56355-7.
[78] S. Idris, T. Karunathilake, and A. Förster, "Survey and Comparative Study of LoRa-Enabled Simulators for Internet of Things and Wireless Sensor Networks," *Sensors 2022, Vol. 22, Page 5546*, vol. 22, no. 15, p. 5546, Jul. 2022, doi: 10.3390/S22155546.

[79] D. H. Wang and W. H. Liao, “Wireless transmission for health monitoring of large structures,” *IEEE Trans Instrum Meas*, vol. 55, no. 3, pp. 972–981, 2006, doi: 10.1109/TIM.2006.873801.

[80] X. Wang, H. Ge, W. Zhang, and Y. Li, “Design of elevator running parameters remote monitoring system based on Internet of Things,” *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, vol. 2015-Novem, pp. 549–555, 2015, doi: 10.1109/ICSESS.2015.7339118.

[81] R. Satapathy, S. Prahlad, and V. Kaulgud, “Smart Shelfie - Internet of shelves: For higher on-shelf availability,” in *Proceedings - 2015 IEEE Region 10 Symposium, TENSYMP 2015*, 2015, pp. 70–73. doi: 10.1109/TENSYMP.2015.9.

[82] C. Swedberg, “ShelfX Unveils Store Shelves for Automating Purchases,” *RFID Journal*, 2011.

[83] T. Jung, N. Jazdi, and M. Weyrich, “A survey on dynamic simulation of automation systems and components in the internet of things,” in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, IEEE, Sep. 2017, pp. 1–4. doi: 10.1109/ETFA.2017.8247770.

[84] R. M. Fujimoto, “Parallel discrete event simulation,” *Commun ACM*, vol. 33, no. 10, pp. 30–53, Oct. 1990, doi: 10.1145/84537.84545.

[85] T. J. Schriber, D. T. Brunner, and J. S. Smith, “Inside discrete-event simulation software: How it works and why it matters,” in *Proceedings - Winter Simulation Conference*, [Piscataway, N.J.]: IEEE, 2015, pp. 132–146. doi: 10.1109/WSC.2014.7019884.

[86] G. S. Fishman, *Discrete-event simulation*, 1., st Edi. New York, NY: Springer New York, 2013.

[87] D. Matloff, Norm S (University of California, “Introduction to discrete-event simulation and the simpy language,” *Davis, CA. Dept of Computer Science. University*, pp. 1–33, 2008, doi: 10.1007/978-0-387-68612-7_10.

[88] U. Hedtstück, *Ereignisorientierte Simulation diskreter Prozesse*. Berlin [u.a.]: Springer, 2013. doi: 10.1007/978-3-642-34871-6_5.

[89] R. Meyer, “Agenten in Raum und Zeit Diskrete Simulation mit Multiagentensystemen und expliziter Raumrepräsentation,” Dec. 2014.

[90] X. Chang, “Network simulations with OPNET,” in *Winter Simulation Conference Proceedings*, New York, NY: ACM, 1999, pp. 307–316. doi: 10.1145/324138.324232.

[91] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli, “A Simulation Platform for Large-Scale Internet of Things Scenarios in Urban Environments,” 2014. doi: 10.4108/icst.urb-iot.2014.257268.

[92] S. N. Han *et al.*, *DPWSim: A simulation toolkit for IoT applications using devices profile for web services*. Piscataway, NJ: IEEE, 2014. doi: 10.1109/WF-IoT.2014.6803226.

[93] X. Zeng, S. K. Garg, P. Strazdins, P. P. Jayaraman, D. Georgakopoulos, and R. Ranjan, “IOTSim: A simulator for analysing IoT applications,” *Journal of Systems Architecture*, vol. 72, pp. 93–107, 2017, doi: 10.1016/j.sysarc.2016.06.008.

[94] V. Looga, Z. Ou, Y. Deng, and A. Yla-Jaaski, “MAMMOTH: A massive-scale emulation platform for Internet of Things,” in *Proceedings - 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems, IEEE CCIS 2012*, [Piscataway, N.J.]: IEEE, Oct. 2013, pp. 1235–1239. doi: 10.1109/CCIS.2012.6664581.

[95] Y. Song, B. Han, X. Zhang, and D. Yang, “Modeling and simulation of smart home scenarios based on Internet of Things,” in *Proceedings - 2012 3rd IEEE International Conference on Network Infrastructure and Digital Content, IC-NIDC 2012*, [Piscataway, N.J.: IEEE, Sep. 2012, pp. 596–600. doi: 10.1109/ICNIDC.2012.6418824.

[96] L. Liu, F. Felgner, G. Frey, and A.--L. für Automatisierungstechnik, “Modellierung und Simulation von Cyber-Physical Systems,” *Proceedings of the 12th Fachtagung EntwurfkomplexerAutomatisierungssysteme (EKA 2012)*, pp. 149–157, 2012.

[97] M. Kirsche, “Simulating the internet of things in a hybrid way,” *Proceedings of the Networked Systems (NetSys) 2013 PhD Forum*, vol. 3, no. NetSys, pp. 1–2, 2013.

[98] S. Sotiriadis, N. Bessis, E. Asimakopoulou, and N. Mustafee, “Towards simulating the internet of things,” in *Proceedings - 2014 IEEE 28th International Conference on Advanced Information Networking and Applications Workshops, IEEE WAINA 2014*, S. Sotiriadis, N. Bessis, E. Asimakopoulou, and N. Mustafee, Eds., IEEE, May 2014, pp. 444–448. doi: 10.1109/WAINA.2014.74.

[99] S. Arthur, H. Li, and R. Lark, “The emulation and simulation of internet of things devices for building information modelling (BIM),” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, Jun. 2018, pp. 325–338. doi: 10.1007/978-3-319-91638-5_18.

[100] J. R. E. Leite, E. L. Ursini, and P. S. Martins, “Performance analysis of iot networks with mobility via modeling and simulation,” in *Simulation Series*, The Society for Modeling and Simulation International, 2018, pp. 35–47. doi: 10.1109/SPECTS.2018.8574144.

[101] M. Chen, Y. Miao, I. Humar, M. Chen, Y. Miao, and I. Humar, “Simulation of Narrowband Cellular Internet of Things,” in *OPNET IoT Simulation*, Springer Singapore, 2019, pp. 605–630. doi: 10.1007/978-981-32-9170-6_10.

[102] Y. Miao, W. Li, D. Tian, M. S. Hossain, and M. F. Alhamid, “Narrowband internet of things: Simulation and modeling,” *IEEE Internet Things J*, vol. 5, no. 4, pp. 2304–2314, Aug. 2018, doi: 10.1109/JIOT.2017.2739181.

[103] A. Borshchev and A. Filippov, “From System Dynamics to Agent Based Modeling,” in *Simulation*, vol. 66, no. 11, 2004, pp. 25–29.

[104] C. O. Franziska Klügl, “Multi-Agent Modelling in Comparison to Standard Modelling,” *AIS*, 2002.

[105] C. M. MacAl and M. J. North, “Tutorial on agent-based modelling and simulation,” *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010, doi: 10.1057/jos.2010.3.

[106] X. Chen, Y. S. Ong, P. S. Tan, N. S. Zhang, and Z. Li, “Agent-based modeling and simulation for supply chain risk management - A survey of the state-ofthe- Art,” in *Proceedings - 2013 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2013*, Piscataway, NJ: IEEE, 2013, pp. 1294–1299. doi: 10.1109/SMC.2013.224.

[107] F. Klügl, “Multiagentensimulation,” *Informatik-Spektrum*, vol. 29, no. 6, pp. 412–415, Dec. 2006, doi: 10.1007/s00287-006-0115-7 M4 - Citavi.

[108] C. M. Macal and M. J. North, “Agent-based modeling and simulation.” Winter Simulation Conference, 2009.

[109] R. Herrler, F. Puppe, and W. Lamersdorf, “Agentenbasierte Simulation zur Ablaufoptimierung in Krankenhäusern und anderen verteilten, dynamischen Umgebungen,” *Fakult¨at f ¨ur Mathematik und Informatik*. 2007.

[110] E. Bonabeau, “Agent-based modeling: Methods and techniques for simulating human systems,” *Proc Natl Acad Sci U S A*, vol. 99, no. SUPPL. 3, pp. 7280–7287, May 2002, doi: 10.1073/pnas.082080899.

[111] W. U. Raffel, “Agentenbasierte Simulation als Verfeinerung der Diskreten-Ereignis-Simulation unter besonderer Berücksichtigung des Beispiels Fahrerloser Transportsysteme,” {Freie Universität Berlin} and {Freie Universität Berlin, Germany}, 2005. [Online]. Available: http://www.diss.fu-berlin.de/diss/servlets/MCRFileNodeServlet/FUDISS_derivate_000000001542/00_RaffelInhalt.pdf?hosts=

[112] C. Macal and M. North, “Introductory tutorial: agent-based modeling and simulation.” IEEE Press, 2014.

[113] G. Fortino, W. Russo, and C. Savaglio, “Agent-oriented modeling and simulation of IoT networks,” in *Proceedings of the 2016 Federated Conference on Computer Science and*

Information Systems, FedCSIS 2016, in Annals of Computer Science and Information Systems. IEEE, 2016, pp. 1449–1452. doi: 10.15439/2016F359.

[114] S. Karnouskos and M. M. J. Tariq, "An agent-based simulation of SOA-ready devices," in *Proceedings - UKSim 10th International Conference on Computer Modelling and Simulation, EUROSIM/UKSim2008*, 2008, pp. 330–335. doi: 10.1109/UKSIM.2008.81.

[115] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi, "CupCarbon: A multi-agent and discrete event Wireless Sensor Network design and simulation tool," in *SIMUTools 2014 - 7th International Conference on Simulation Tools and Techniques*, F. Barros, K. Perumalla, and R. Roland, Eds., 2014, pp. 126–131. doi: 10.4108/icst.simutools.2014.254811.

[116] M. Dyk, A. Najgebauer, and D. Pierzchala, *SenseSim: An agent-based and discrete event simulator for Wireless Sensor Networks and the Internet of Things*. Piscataway, NJ; Piscataway, NJ: IEEE, 2015. doi: 10.1109/WF-IoT.2015.7389078.

[117] S. Karnouskos and T. N. De Holanda, "Simulation of a smart grid city with software agents," in *EMS 2009 - UKSim 3rd European Modelling Symposium on Computer Modelling and Simulation*, 2009, pp. 424–429. doi: 10.1109/EMS.2009.53.

[118] G. D'Angelo, S. Ferretti, and V. Ghini, *Simulation of the Internet of Things*. Piscataway, NJ: IEEE, 2016. doi: 10.1109/HPCSim.2016.7568309.

[119] S. Karnouskos and M. M. J. Tariq, "Using multi-agent systems to simulate dynamic infrastructures populated with large numbers of web service enabled devices," in *Proceedings - 2009 International Symposium on Autonomous Decentralized Systems, ISADS 2009*, 2009, pp. 209–216. doi: 10.1109/ISADS.2009.5207354.

[120] L. J. Perez and J. S. Rodriguez, "Simulation of scalability in IoT applications," in *International Conference on Information Networking*, IEEE Computer Society, Apr. 2018, pp. 577–582. doi: 10.1109/ICOIN.2018.8343184.

[121] S. Bosmans, S. Mercelis, J. Denil, and P. Hellinckx, "Testing IoT systems using a hybrid simulation based testing approach," *Computing*, vol. 101, no. 7, pp. 857–872, Jul. 2019, doi: 10.1007/s00607-018-0650-5.

[122] G. Fortino, R. Gravina, W. Russo, and C. Savaglio, "Modeling and Simulating Internet-of-Things Systems: A Hybrid Agent-Oriented Approach," *Comput Sci Eng*, vol. 19, no. 5, pp. 68–76, 2017, doi: 10.1109/MCSE.2017.3421541.

[123] C. Scheifele, O. Riedel, and A. Verl, "Virtuelle Inbetriebnahme komplexer Produktionsanlagen mittels echtzeitfähiger Co-Simulation," *VDI-Bericht 2293 Automation 2017*, 2017.

[124] C. Gomes, C. Thule, D. Broman, P. G. Larsen, and H. Vangheluwe, "Co-simulation: State of the art," *ArXiv*, 2017.

[125] R. Schmoll, *Co-Simulation und Solverkopplung*, vol. 3/2015. Kassel, Hess: Kassel University Press, 2015.

[126] M. Stifter, E. Widl, F. Andren, A. Elsheikh, T. Strasser, and P. Palensky, "Co-simulation of components, controls and power systems based on open source software," in *IEEE Power and Energy Society General Meeting*, Piscataway, NJ: IEEE, 2013, pp. 1–5. doi: 10.1109/PESMG.2013.6672388.

[127] C. Farhat and M. Lesoinne, "Two efficient staggered algorithms for the serial and parallel solution of three-dimensional nonlinear transient aeroelastic problems," *Comput Methods Appl Mech Eng*, vol. 182, no. 3–4, pp. 499–515, 2000, doi: 10.1016/S0045-7825(99)00206-6.

[128] S. Schulte, *Modulare und hierarchische Simulation gekoppelter Probleme*, Als Ms. ge., vol. Nr. 271. in Fortschritt-Berichte VDI : Reihe 20, Rechnerunterstützte Verfahren, vol. Nr. 271. Düsseldorf: VDI-Verl., 1998.

[129] M. Benedikt, J. Zehetner, D. Watzenig, and J. Bernasch, “Moderne Kopplungsmethoden - Ist Co-Simulation beherrschbar?,” *NAFEMS Online-Magazin*, vol. 22, no. 2/2012, pp. 63–74, 2012.
[130] R. Kübler and W. Schiehlen, “Two Methods of Simulator Coupling,” *Math Comput Model Dyn Syst*, vol. 6, no. 2, pp. 93–113, 2000, doi: 10.1076/1387-3954(200006)6:2;1-M;FT093.
[131] Martin. Busch, “Zur Effizienten Kopplung von Simulationsprogrammen,” Dissertation, Universität Kassel, 2012.
[132] D. Peshev and A. G. Livingston, “OSN Designer, a tool for predicting organic solvent nanofiltration technology performance using Aspen One, MATLAB and CAPE OPEN,” *Chem Eng Sci*, vol. 104, pp. 975–987, 2013, doi: 10.1016/j.ces.2013.10.033.
[133] M. Oppelt, G. Wolf, and L. Urbas, “Capability-analysis of co-simulation approaches for process industries,” in *19th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2014*, A. Grau and H. Martinez, Eds., [Piscataway, N.J.]: IEEE, 2014, pp. 1–4. doi: 10.1109/ETFA.2014.7005292.
[134] K. Hopkinson, X. Wang, R. Giovanini, J. Thorp, K. Birman, and D. Coury, “EPOCHS,” *IEEE Transactions on Power Systems*, vol. 21, no. 2, pp. 548–558, 2006, doi: 10.1109/TPWRS.2006.873129 T4 - A Platform for Agent-Based Electric Power and Communication Simulation Built From Commercial Off-the-Shelf Components M4 - Citavi.
[135] S. Schütte, S. Scherfke, and M. Tröschel, “Mosaik: A framework for modular simulation of active components in Smart Grids,” in *2011 IEEE 1st International Workshop on Smart Grid Modeling and Simulation, SGMS 2011*, [Piscataway, N.J.]: IEEE, 2011, pp. 55–60. doi: 10.1109/SGMS.2011.6089027.
[136] J. Nutaro, P. T. Kuruganti, L. Miller, S. Mullen, and M. Shankar, “Integrated hybrid-simulation of electric power and communications systems,” in *2007 IEEE Power Engineering Society General Meeting, PES*, Piscataway, N.J.: IEEE Xplore, 2007, pp. 1–8. doi: 10.1109/PES.2007.386202.
[137] T. Blockwitz *et al.*, “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*, in Link?ping Electronic Conference Proceedings, vol. 76. Link?ping University Electronic Press, 2012, pp. 173–184. doi: 10.3384/ecp12076173.
[138] C. Bertsch, E. Ahle, and U. Schulmeister, “The Functional Mockup Interface - seen from an industrial perspective,” in *Proceedings of the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden*, in Linköping Electronic Conference Proceedings, vol. 96. Linköping University Electronic Press, 2014, pp. 27–33. doi: 10.3384/ecp1409627.
[139] J. Bastian, C. Clauß, S. Wolf, and P. Schneider, “Master for Co-Simulation Using FMI,” in *Proceedings from the 8th International Modelica Conference, Technical Univeristy, Dresden, Germany*, in Linköping Electronic Conference Proceedings, vol. 63. Linköping University Electronic Press, 2011, pp. 115–120. doi: 10.3384/ecp11063115.
[140] R. Fujimoto, *Parallel and distributed simulation*, vol. 2016-Febru. in Wiley series on parallel and distributed computing, vol. 2016- Febru. New York [etc.]: John Wiley & Sons, 2016. doi: 10.1109/WSC.2015.7408152.
[141] Simulation Interoperability Standards Committee (SISC), *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*. Institute of Electrical and Electronics Engineers, 2000. doi: 10.1109/IEEESTD.2000.92296.
[142] Pitch, “The HLA Tutorial: A Practical Guide for Developing Fistributed Simulations,” p. 5, 2012.
[143] G. Lasnier, J. Cardoso, P. Siron, C. Pagetti, and P. Derler, “Distributed simulation of heterogeneous and real-time systems,” in *Proceedings - IEEE International Symposium*

on Distributed Simulation and Real-Time Applications, A. Verbraeck, Ed., Piscataway, NJ: IEEE, 2013, pp. 55–62. doi: 10.1109/DS-RT.2013.14.

[144] T. Nagele and J. Hooman, "Co-simulation of cyber-physical systems using HLA," in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference, CCWC 2017*, S. Chakrabarti and H. N. Saha, Eds., Piscataway, NJ: IEEE, 2017, pp. 1–6. doi: 10.1109/CCWC.2017.7868401.

[145] T. Nägele and J. Hooman, "Scalability Analysis of Cloud-Based Distributed Simulations of IoT Systems Using HLA," in *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, IEEE Computer Society, Feb. 2019, pp. 1075–1080. doi: 10.1109/PADSW.2018.8644925.

[146] J. Kolsch, A. Ratzke, and C. Grimm, "Co-simulating the internet of things in a smart grid use case scenario," in *7th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, MSCPES 2019 - Held as part of CPS Week, Proceedings*, Institute of Electrical and Electronics Engineers Inc., Apr. 2019. doi: 10.1109/MSCPES.2019.8738795.

[147] O. Palizban and K. Kauhaniemi, "Multi-Simulation Environment for Smart Grid: Co-simulation Approach," in *2nd International Conference on Smart Grid and Renewable Energy, SGRE 2019 - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Nov. 2019. doi: 10.1109/SGRE46976.2019.9020905.

[148] X. Li, Q. Huang, and D. Wu, "Distributed Large-Scale Co-Simulation for IoT-Aided Smart Grid Control," *IEEE Access*, vol. 5, pp. 19951–19960, Sep. 2017, doi: 10.1109/ACCESS.2017.2753463.

[149] S. Mittal, A. Tolk, A. Pyles, N. Van Balen, and K. Bergollo, "Digital Twin Modeling, Co-Simulation and Cyber Use-Case Inclusion Methodology for IOT Systems," in *Proceedings - Winter Simulation Conference*, Institute of Electrical and Electronics Engineers Inc., Dec. 2019, pp. 2653–2664. doi: 10.1109/WSC40007.2019.9004656.

[150] O. P. C. Foundation, "OPC-UA Specification".

[151] S. Hensel, M. Graube, L. Urbas, T. Heinzerling, and M. Oppelt, "Co-simulation with OPC UA," *IEEE International Conference on Industrial Informatics (INDIN)*, vol. 0. IEEE, Piscataway, NJ, pp. 20–25, 2016. doi: 10.1109/INDIN.2016.7819127.

[152] T. Miettinen, "Synchronized Cooperative Simulation : OPC UA Based Approach," p. 107, 2012.

[153] The Osgi Alliance, "OSGi Core," no. June, 2014.

[154] J. McAffer, P. VanderLei, and S. J. Archer, *OSGi and Equinox*. Upper Saddle River: Addison-Wesley, 2010. [Online]. Available: http://www.worldcat.org/oclc/781030277

[155] M. Oppelt, O. Drumm, B. Lutz, and G. Wolf, "Approach for integrated simulation based on plant engineering data," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, C. Seatzu, Ed., Piscataway, NJ: IEEE, Sep. 2013, pp. 1–4. doi: 10.1109/ETFA.2013.6648156.

[156] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. 2007. doi: 10.1002/9780470058411.

[157] M. Wooldridge and N. R. Jennings, "Agent theories, architectures, and languages: A survey," in *Intelligent agents*, vol. 890, M. J. Wooldridge and N. R. Jennings, Eds., in Lecture Notes in Computer Science, vol. 890. , Berlin [etc.]: Springer-Verlag, pp. 1–39. doi: 10.1007/3-540-58855-8.

[158] N. R. Jennings, "Agent-Oriented Software Engineering," in *Multiple approaches to intelligent systems*, vol. 1611, I. Imam, Ed., in Lecture Notes in Computer Science, vol. 1611. , Berlin [etc.]: Springer, 1999, pp. 4–10. doi: 10.1007/978-3-540-48765-4.

[159] HS Nwana, "Software agents: An overview," *The Knowledge Engineering Review , Volume 11 , Issue 3*, 1996.

[160] G. Roth and P. Senge, “From Theory to Practice From Theory to Practice :,” *Readings in Planning Theory*, vol. 89, no. 10. pp. 1321–1323, 1995. doi: 10.1097/ACM.0000000000000324.
[161] Foundation For Intelligent Physical Agents, “FIPA Agent Management Specification,” *Online*, no. SC00023. 2002.
[162] Foundation For Intelligent Physical Agents, “FIPA ACL Message Structure Specification,” *Online*, no. SC00061G. p. 11, 2002.
[163] F. Schorr and L. Hvam, “Design science research: A suitable approach to scope and research it service catalogs,” in *Proceedings - 2018 IEEE World Congress on Services, SERVICES 2018*, Institute of Electrical and Electronics Engineers Inc., Oct. 2018, pp. 27–28. doi: 10.1109/SERVICES.2018.00026.
[164] K. Peffers, M. Rothenberger, and B. Kuechler, Eds., *Design Science Research in Information Systems. Advances in Theory and Practice*, vol. 7286. in Lecture Notes in Computer Science, vol. 7286. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-29863-9.
[165] T. Trepper and T. Trepper, “Forschungsmethodik,” in *Fundierung der Konstruktion agiler Methoden*, Springer Fachmedien Wiesbaden, 2015, pp. 11–28. doi: 10.1007/978-3-658-10090-2_2.
[166] A. Hevner and S Chatterjee, “Design Science Research in Information Systems,” *Design research in information - Springer*, pp. 9–22, 2010, doi: 10.1007/978-1-4419-5653-8_2.
[167] S. Gregor and AR Hevner, “Positioning and presenting design science research for maximum impact,” *MIS Quarterly*, vol. June, 2013.
[168] R. Wieringa, “Design science methodology for information systems and software engineering,” Dec. 2014.
[169] T. Jung, P. Shah, and M. Weyrich, “Dynamic Co-Simulation of Internet-of-Things-Components using a Multi-Agent-System,” *Procedia CIRP*, vol. 72, pp. 874–879, 2018, doi: 10.1016/j.procir.2018.03.084.
[170] M. W. Tobias Jung, Nasser Jazdi, “Dynamische Co-Simulation von Automatisierungssystemen und ihren Komponenten im Internet der Dinge Prozessorientierte Interaktion von IoT-Komponenten,” in *15. Fachtagung EKA – Entwurf komplexer Automatisierungssysteme*, 2018.
[171] M. W. Tobias Jung, Nasser Jazdi, “Dynamische Co-Simulation von Automatisierungssystemen und ihren Komponenten im Internet der Dinge,” in *19. VDI-Leitkongress „AUTOMATION 2018“*, 2018.
[172] T. Jung and M. Weyrich, “Synchronization of a ‘plug-and-Simulate’-capable Co-Simulation of Internet-of-Things-Components,” in *Procedia CIRP*, 2019, pp. 367–372. doi: 10.1016/j.procir.2019.02.090.
[173] Modelica Association Project “FMI,” “Functional Mock-up Interface for Model Exchange and Co-Simulation,” no. 07006, pp. 1–120, 2013.
[174] O. Topçu and H. Oğuztüzün, *Guide to Distributed Simulation with HLA*. in Simulation Foundations, Methods and Applications. Cham, Switzerland: Springer, 2017. doi: 10.1007/978-3-319-61267-6.
[175] IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999), “802.11ah-2016 - IEEE Standard for Information technology--Telecommunications and information exchange between systems - Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY,” *30 April 2017*, no. February, p. 594, 2017, doi: 10.1109/IEEESTD.2017.7920364.
[176] D. ISO, “DIN EN ISO 10303,” 2008.
[177] R. M. Fujimoto, “Time Management in The High Level Architecture,” *Simulation*, vol. 71, no. 6, pp. 388–400, Dec. 1998, doi: 10.1177/003754979807100604.

[178] S. Jafer, Q. Liu, and G. Wainer, “Synchronization methods in parallel and distributed discrete-event simulation,” *Simul Model Pract Theory*, vol. 30, pp. 54–73, Jan. 2013, doi: 10.1016/j.simpat.2012.08.003.
[179] D. R. Jefferson, “Virtual Time,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 7, no. 3, pp. 404–425, Jul. 1985, doi: 10.1145/3916.3988.
[180] IEEE, “1516-2010 - IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA),” *IEEE Std 1516-2010 (Revision of IEEE Std 1516-2000)*. IEEE, Piscataway, NJ, USA, pp. 1–38, 2010. doi: 10.1109/IEEESTD.2010.5553440.
[181] “Jadex Active Components 3.0.1 Documentation.” Accessed: Apr. 18, 2020. [Online]. Available: https://download.actoron.com/docs/releases/jadex-3.0.1/jadex-mkdocs/
[182] J. C. Mankins, “Technology Readiness Level,” 2004.
[183] M. W. Tobias Jung, Nasser Jazdi, Stefan Krauß, Christian Köllner, “Hardware-in-the-Loop Simulation for a Dynamic Co-Simulation of Internet-of-Things-Components,” in *53rd CIRP Conference on Manufacturing Systems*, 2020.

Anhang

Erweiterung durch Hardware-in-the-Loop

Wie schon in der Einleitung in Kapitel 1.1 erwähnt, werden auch Hardware-in-the-Loop (HiL) Simulationen immer wichtiger. Hierbei wird beispielsweise die Umgebung einer Steuerung simuliert, um eine real vorhandene Steuerung testen und in Betrieb nehmen zu können. Dieses Prinzip ist auch für IoT-Komponenten interessant, so dass IoT-Komponenten mit einer sogenannten Rest-IoT-Simulation getestet werden können. Bei einer Rest-IoT-Simulation wird das restliche IoT, also das IoT-System ohne die zu testende Komponente simuliert. Für die zu testende Komponente sieht es so aus, als ob sie sich in einem IoT-System befindet. Gerade hierbei ist es wichtig, dass zur Laufzeit ein Ein- und Austreten von Komponenten simuliert werden kann. Daher wird untersucht, ob sich das Konzept der „Plug-and-Simulate“-fähigen Co-Simulation auf Hardware-in-the-Loop Simulationen anwenden lässt. Dies ist als reine Erweiterung des bestehenden Konzeptes zu sehen und nicht als Kernbestandteil.

Um HiL-Simulationen grundsätzlich zu ermöglichen, muss die zu testende Komponente ebenfalls durch einen Simulationsvertreter über eine zu implementierende Schnittstelle in die Co-Simulation eingebunden werden (siehe Abbildung). Dadurch ist diese Komponente ebenfalls gekapselt und stellt sich für die restlichen Teilsimulationen ebenfalls als Teilsimulation dar. Die Komponente selbst erhält durch den Simulationsvertreter und die Schnittstelle die simulierten Eingangssignale als Reaktion auf die Ausgangssignale und kann somit mit den simulierten Komponenten interagieren. Die Umsetzung der Schnittstelle erfolgt gleich wie bei den Komponentensimulationen, es gibt eine vertikale Aufteilung in kommunikationsorientierte und prozessorientierte Schnittstelle und eine horizontale Aufteilung in einen generischen und einen komponentenspezifischen Teil. Eine Synchronisationsschnittstelle wird nicht benötigt, da eine reale Komponente nicht synchronisiert werden kann. Der HiL-Vertreter sieht exakt gleich aus wie ein Komponentenvertreter, mit dem Unterschied, dass er dem Taktgeber nach Ablauf der Länge des Simulationsschritts in realer Zeit zurückmeldet, dass ein Zeitschritt erledigt ist. Aufgrund dieser hohen Übereinstimmung wird im Folgenden nicht zwischen Komponentenvertreter und HiL-Vertreter differenziert. Ein Beispiel einer Implementierung der HiL-Schnittstelle wird weiter unten gegeben.

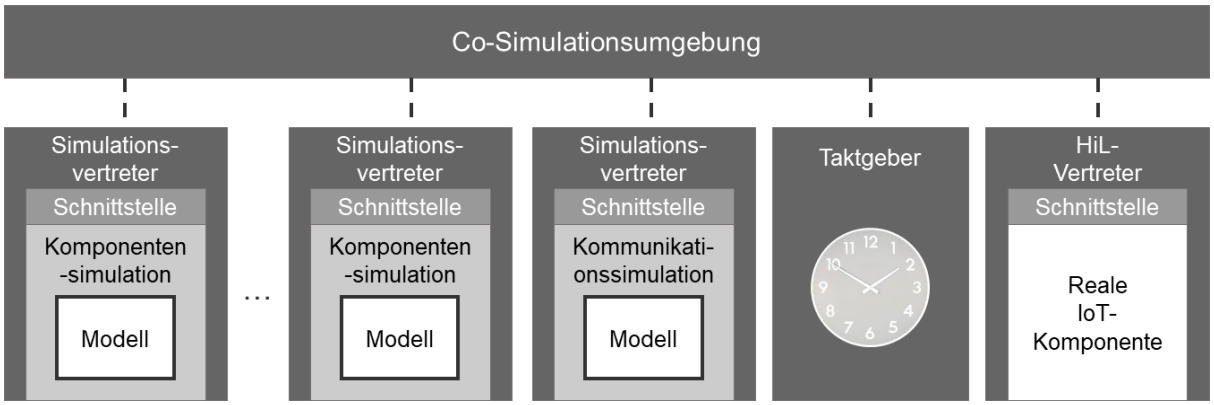


Abbildung: Einbinden einer realen IoT-Komponente als HiL-Simulation

Somit ist eine Hardware-in-the-Loop Simulation grundsätzlich möglich, allerdings ergeben sich zwei Einschränkung. Prozessorientierte Interaktionen, die andere Komponenten beeinflussen, müssen in die Prozesssimulation verlagert werden, da sie sonst nicht die anderen Teilsimulationen erreichen würden. Dies ist aber auch bei simulierten Komponenten der Fall, siehe Kapitel 3.3.4.

Zudem kann die reale Hardwarekomponente nicht synchronisiert werden. Es müssen folglich entweder alle Teilsimulation und alle Interaktionen zwischen diesen in Echtzeit ablaufen oder es können nur zeitunkritische Komponenten in die HiL-Simulation eingebunden werden. Da das bisher präsentierte Konzept Echtzeitanforderungen an die Co-Simulation nicht berücksichtigt (kein Fokus dieser Arbeit), können bisher nur zeitunkritische Komponenten eingebunden werden. Zeitunkritische Komponenten sind Komponenten, die keine schnellen Antworten von der Co-Simulation benötigen, beispielsweise eine Komponente, die, nachdem sie einen Auftrag erhalten hat, diesen abarbeitet und nur zu Beginn der Abarbeitung mit anderen Komponenten interagiert, in 6.1.3 wird ein Beispiel hierfür gegeben.

Für jede Hardwarekomponente muss eine eigene Schnittstelle implementiert werden. In dieser Arbeit wurde ein Raspberry Pi 3 Model B ausgewählt, auf dem eine Steuerung eines Lagerbediengeräts lief. Der Raspberry kommuniziert per WLAN mit dem PC, auf dem die Co-Simulation läuft. Die Nachrichten des Raspberry werden von der implementierten Schnittstelle über einen Socket empfangen und an den HiL-Vertreter weitergeleitet und andersherum.

In der prototypischen Realisierung wurde eine HiL-Simulation mit dem gleichen Ablauf, des in Kapitel 6.1 beschriebenen Szenarios durchgeführt, wobei das Model des Lagerbediengeräts durch die reale Steuerung ersetzt wurde.

Eine ausführliche Beschreibung der Anbindung von HiL-Simulationen findet sich in [183].

Studentische Arbeiten, die bei Untersuchungen in dieser Arbeit unterstützend durchgeführt wurden

Untersuchungen zu Diskreten-Event-Simulationen

D. Di Franco, “Untersuchung und prototypische Umsetzung von Simulationsszenarien im Smart Traffic,” 2017

S. Sivaraman, “Investigation and Prototypical implementation of simulation scenarios for Smart Ware-housing,” 2017

Untersuchungen zu Agentenbasierten Simulationen

J. Li, “Untersuchung und prototypische Umsetzung von agentenbasierten Simulation im Internet der Dinge.” 2017

M. Ragab, “Investigation and prototypical implementation of agent-based simulation of Smart Traffic-scenarios,” 2017

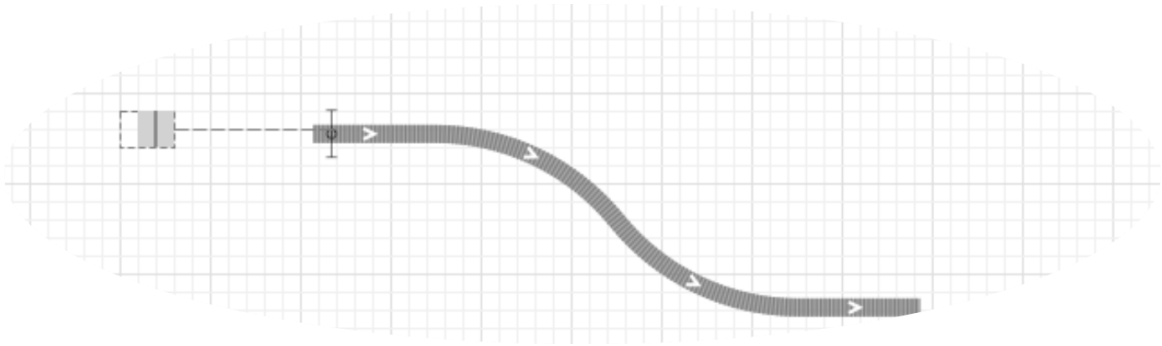
S. Khan, “Investigation and prototypical implementation of agent-based simulation of Smart Ware-house-scenarios,” 2017

Beschreibung der Modelle

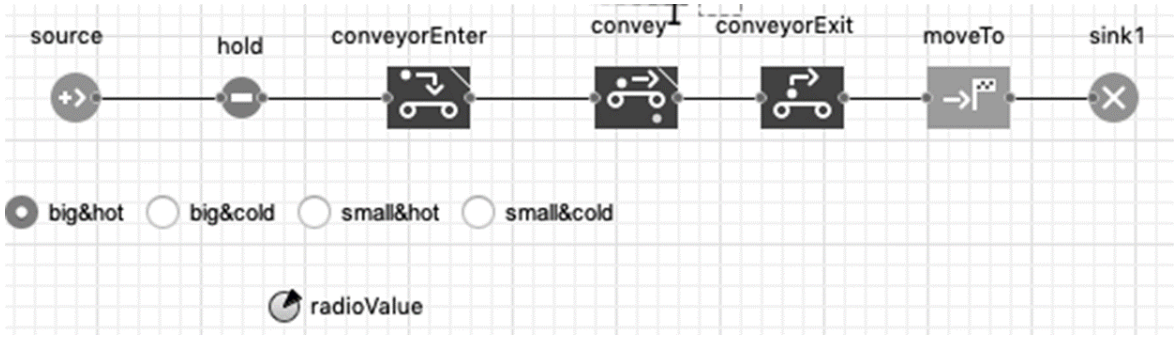
Im Folgenden werden die in 6.1.3 erwähnten Modelle, die in der Evaluierung verwendet wurden dargestellt.

Wareneingang

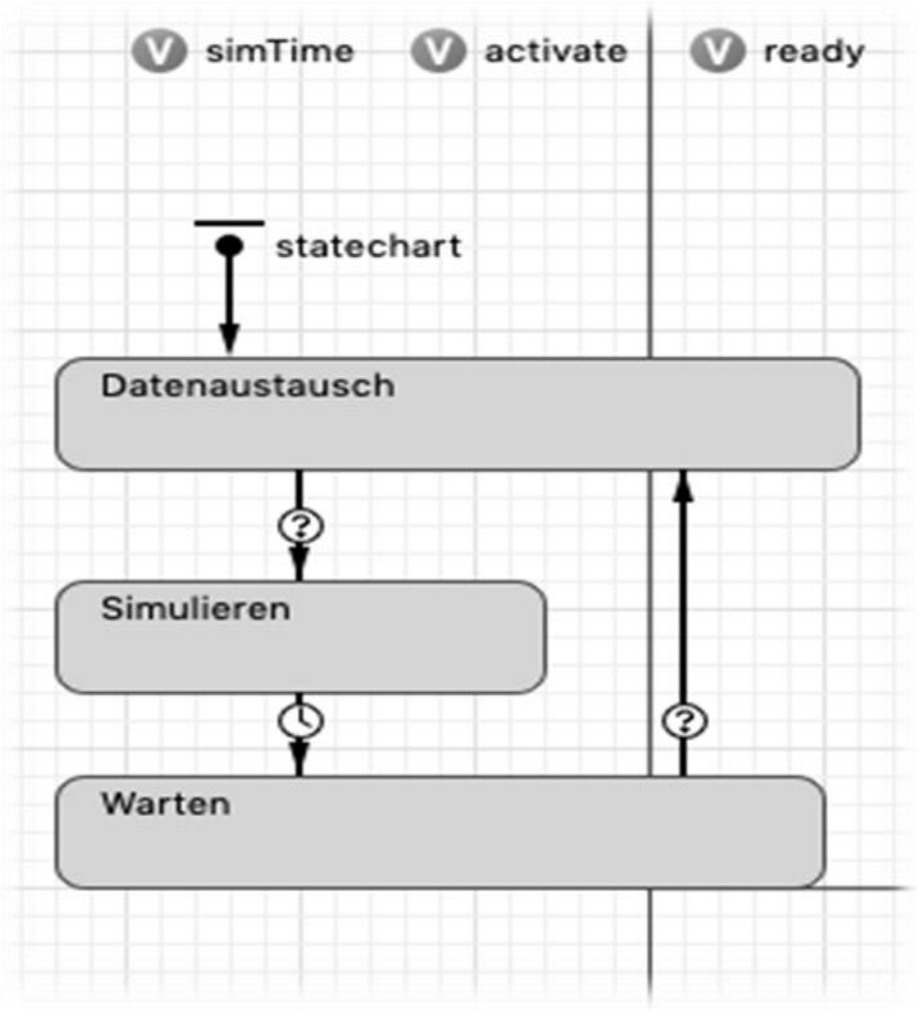
AnyLogic: Modell des Wareneingangs mit unterschiedlichen erzeugten Waren



Ablauf des Wareneingangs

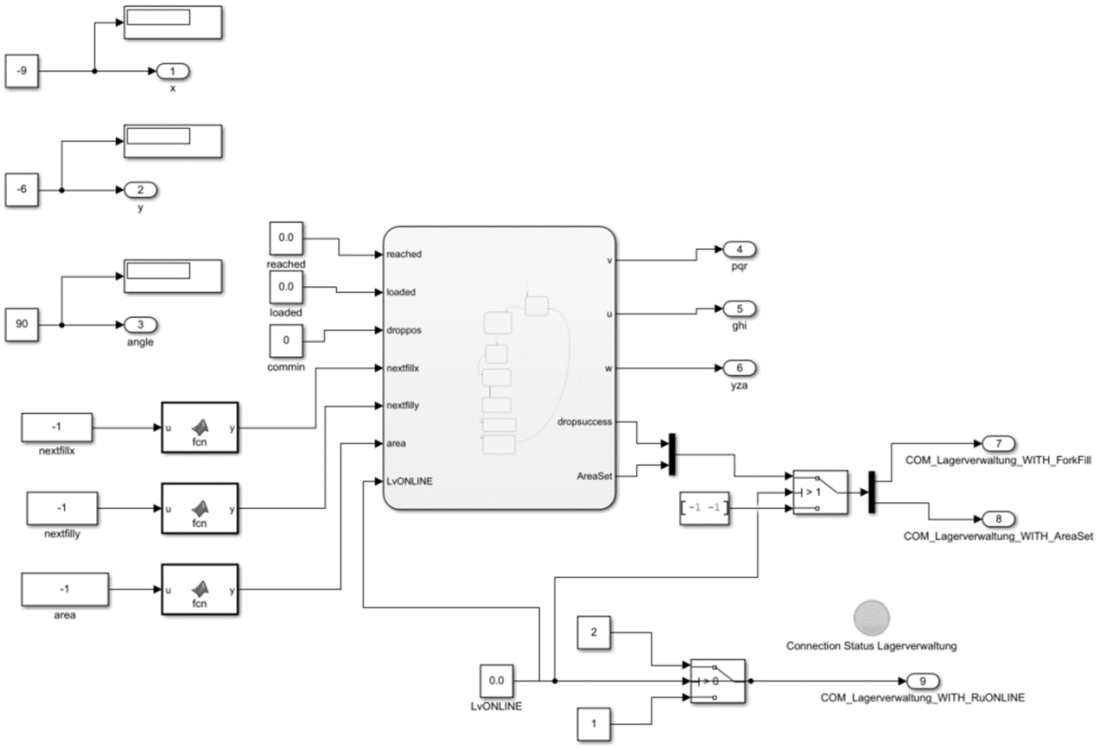


Ablauf des Datenaustauschs und der Synchronisation des Wareneingangs

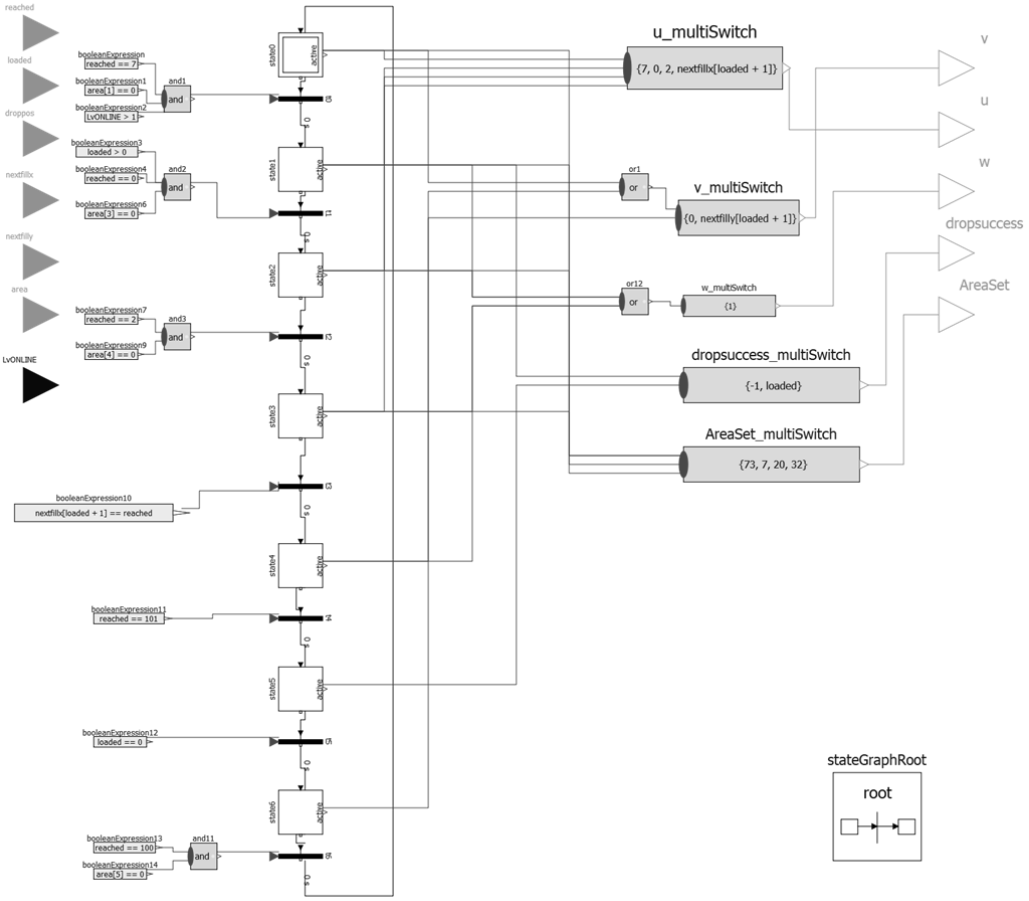


Gabelstapler

Modell des Gabelstaplers in MATLAB Simulink mit Fahrbefehlen und Warentransportbefehlen:

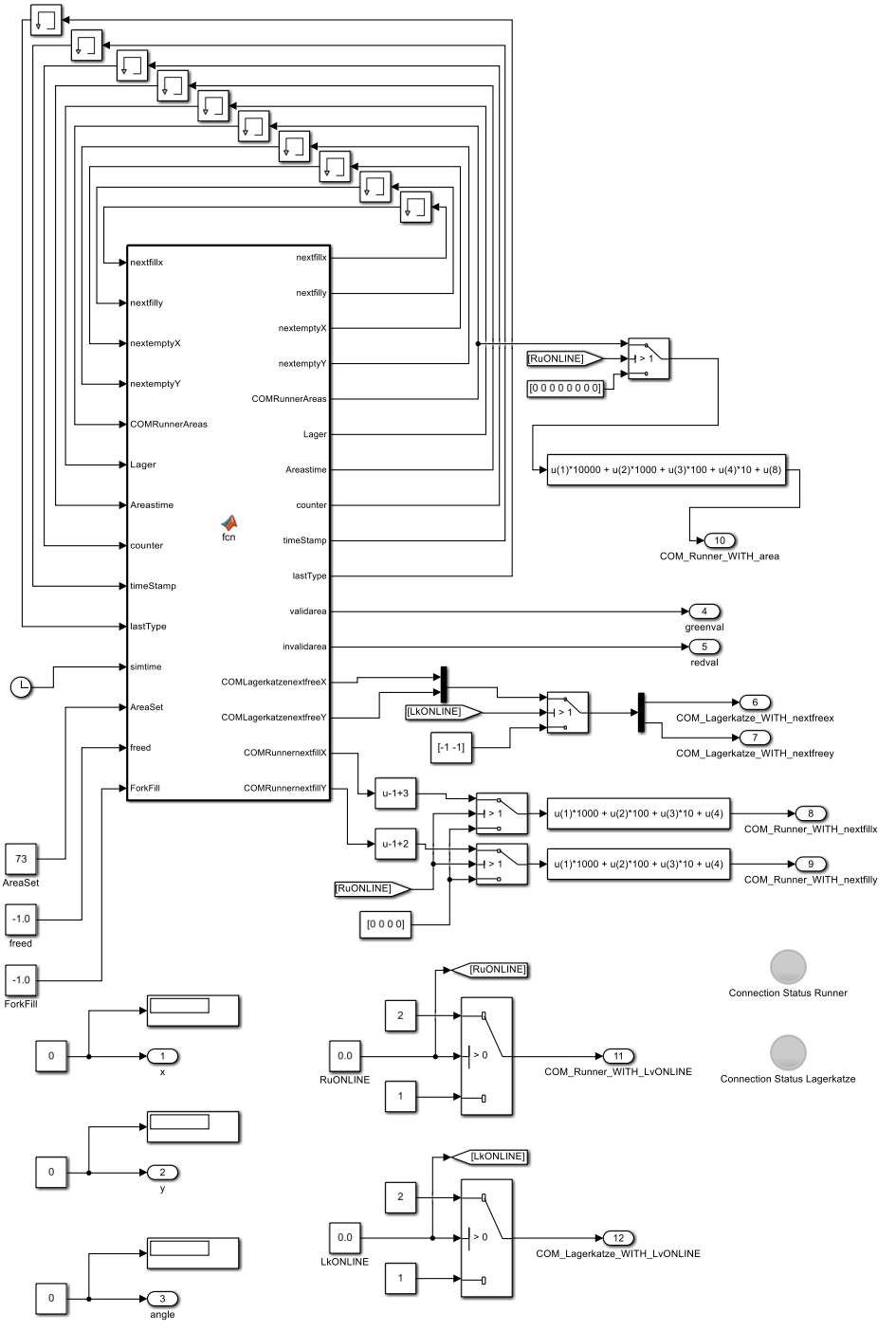


Modell des Gabelstaplers OpenModelica (angebunden über FMI) mit Fahrbefehlen und Warentransportbefehlen:



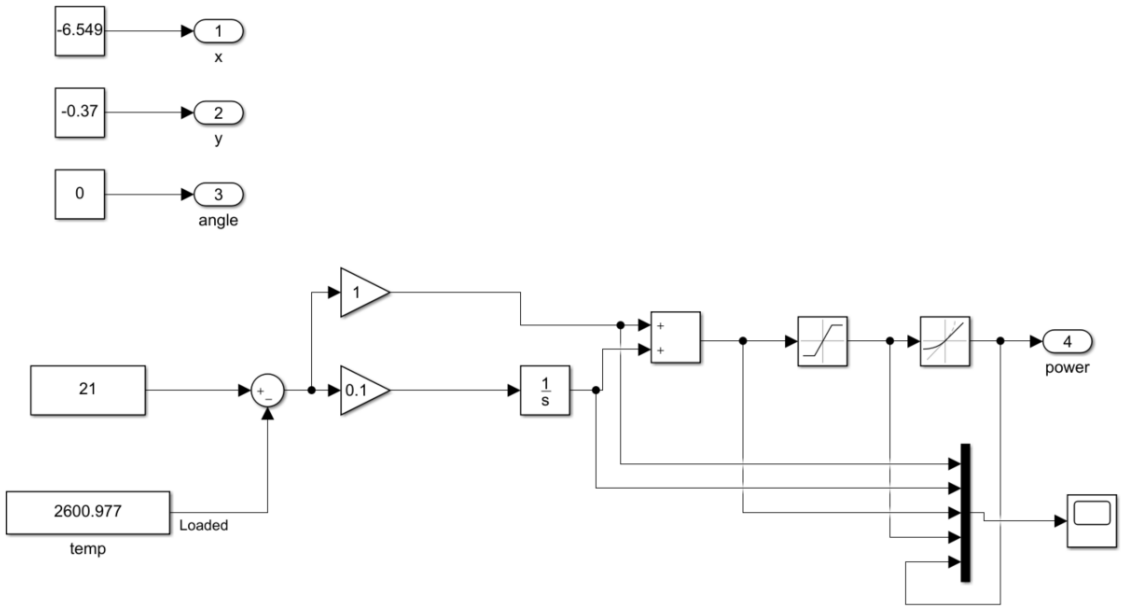
Lagerverwaltung und Lagerregal

Model in MATLAB Simulink (angebunden über FMI), die Lagerverwaltung wird durch das Lagerregal realisiert und managt den gesamten Ablauf des Warenlagers:



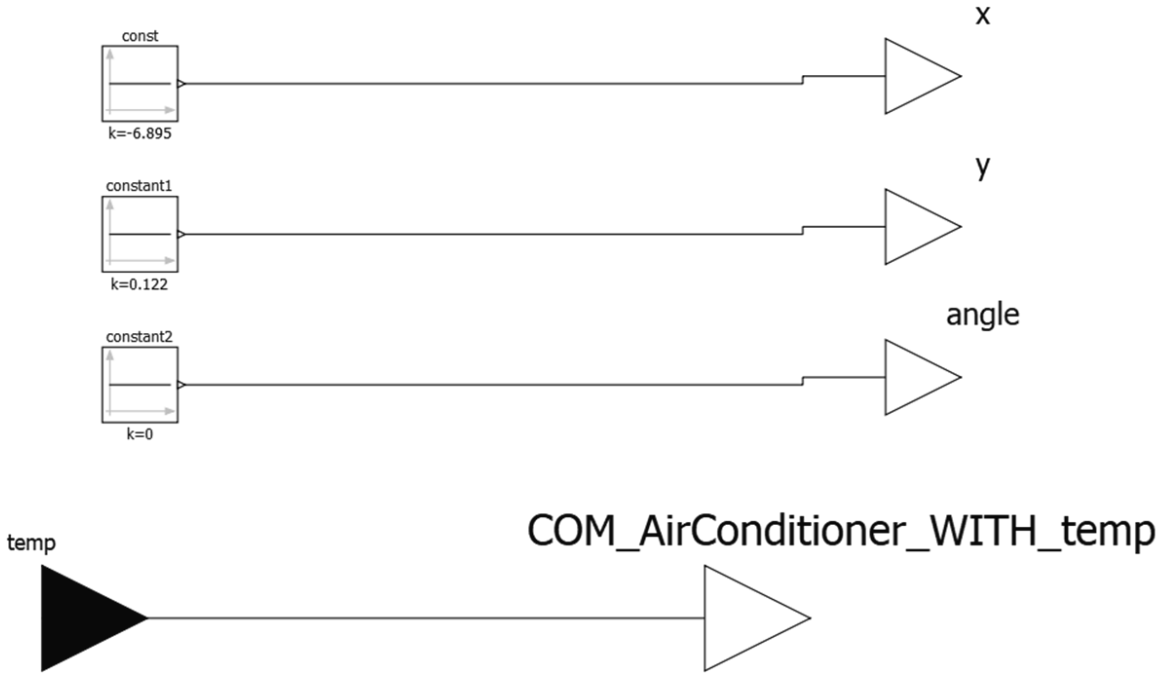
Kühleinheit

Modell der Kühleinheit in MATLAB Simulink (angebunden über FMI):



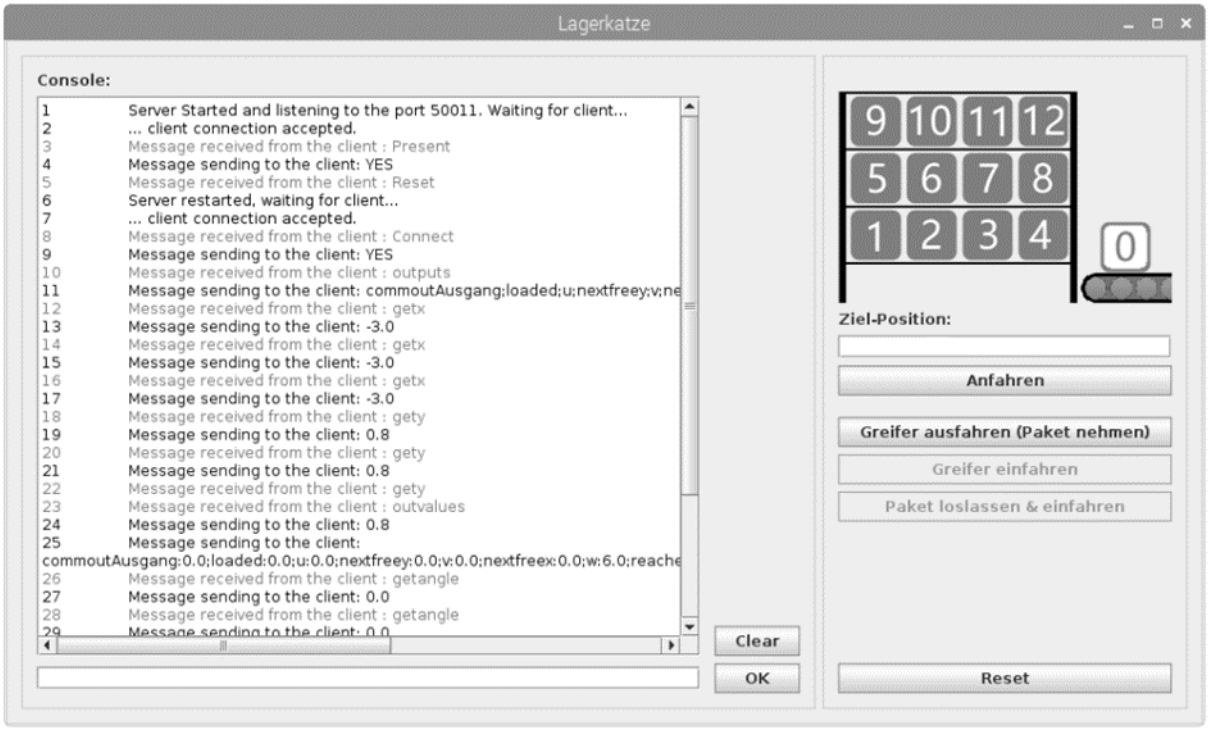
Temperatursensor

Modell des Temperatursensors in OpenModelica (angebunden über FMI):

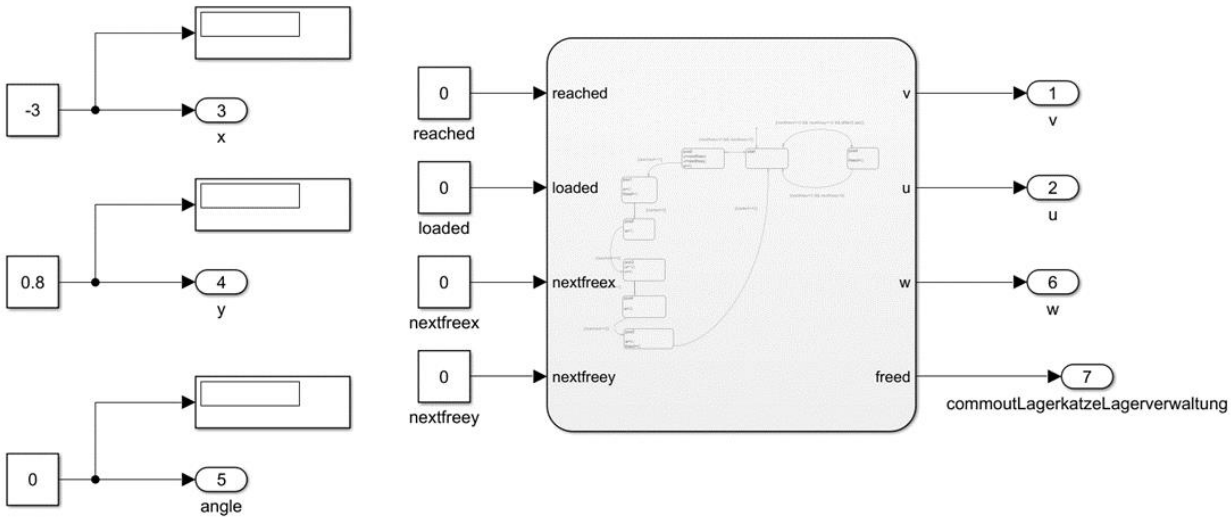


Lagerbediengerät

Benutzeroberfläche der Hardwarekomponente, welche auf einem Raspberry Pi läuft:



Modell der Hardwarekomponente in MATLAB Simulink, es werden die gleichen Funktionen wie auf dem Raspberry Pi erfüllt:



Begriffsverzeichnis

Co-Simulation: wird als eine Simulation verstanden, die aus mehreren Teilsimulationen zusammengesetzt ist. Die Teilsimulationen simulieren Teilaspekte der Gesamtsimulation und werden unter Umständen in verschiedenen Simulationstools ausgeführt.

Dynamik: wird in der vorliegenden Arbeit für das Ein- und Austreten von Komponenten in ein System verwendet.

Digitaler Zwilling: Ein ständig synchrones Abbild der Realität, dass sämtliche Daten, Modelle und Informationen des realen Assets enthält.

Gesamtsimulation: bezeichnet die komplette Co-Simulation. Sie besteht aus mehreren Teilsimulationen, die sich in Komponentensimulationen und Interaktionssimulationen unterteilen.

Heterogenität: bezeichnet das Zusammenwirken von sehr unterschiedlichen Komponenten in einem System.

Interaktionssimulation: ist eine Simulation, die Interaktionen zwischen den Komponentensimulationen simuliert. Sie unterteilen sich in Kommunikationssimulationen und Prozesssimulationen.

Interaktionsvertreter: Ein Interaktionsvertreter ist eine Software, die eine Interaktionssimulation gegenüber der Co-Simulationsumgebung und anderen Teilsimulationen kapselt und vertritt. Kommunikations- und Prozessvertreter sind Interaktionsvertreter.

Internet der Dinge (IoT): bezeichnet die Vernetzung von Dingen untereinander [61].

Kommunikationssimulation: simuliert die Kommunikation des IoT-Systems. Hierüber können somit die einzelnen Kommunikationsnachrichten zwischen den Komponentensimulationen ausgetauscht werden.

Kommunikationsvertreter: ist eine Software, die eine Kommunikationssimulation gegenüber der Co-Simulationsumgebung und anderen Teilsimulationen kapselt und vertritt.

Komponente: ist eine abgeschlossene Einheit, die eine oder mehrere Funktionen erfüllt und über definierte Schnittstellen mit anderen Komponenten oder ihrer Umwelt kommunizieren kann. In dieser Arbeit wird unter dem Begriff Komponente meist eine IoT-Komponente verstanden.

Komponentensimulation: simuliert eine IoT-Komponente und deren logisches Verhalten. Die Interaktionen zwischen den Komponentensimulationen werden durch die Interaktionssimulationen simuliert.

Komponentenvertreter: ist eine Software, die eine Komponentensimulation gegenüber der Co-Simulationsumgebung und anderen Teilsimulationen kapselt und vertritt.

Modell: „*Ein Modell ist eine vereinfachte Nachbildung eines geplanten oder existierenden Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System“* [36]. In dieser Arbeit wird unter diesem Begriff meist ein ausführbares Modell verstanden, also ein Modell, das in einer Simulation verwendet werden kann.

Prozesssimulation: simuliert die physikalischen Prozesse des IoT-Systems. Hierüber können somit die einzelnen physikalischen Interaktionen zwischen den Komponentensimulationen ausgetauscht werden.

Prozessvertreter: ist eine Software, die eine Prozesssimulation gegenüber der Co-Simulationsumgebung und anderen Teilsimulationen kapselt und vertritt.

Plug-and-Simulate: bezeichnet die Eigenschaft von Simulationen, sich in eine Co-Simulation zur Laufzeit zu integrieren, ohne dass Anpassungen an der gesamten Simulation oder an der Teilsimulation vorgenommen werden müssen.

Simulation: „*Verfahren zur Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“* [36].

Simulationsvertreter: ist eine Software, die eine Teilsimulation gegenüber der Co-Simulationsumgebung und anderen Teilsimulationen kapselt und vertritt. Komponenten- und Interaktionsvertreter sind Simulationsvertreter.

Simulationszeit: ist die Zeit, die in der Simulation vergangen ist, dies kann schneller oder langsamer als die real vergangene Zeit sein.

Simulator: „*Softwareprogramm, mit dem ein Modell mithilfe einer Programmiersprache (Simulationssprache) zur Nachbildung des dynamischen Verhaltens eines Systems und seiner Prozesse erstellt und ausführbar gemacht werden kann“* [36].

Teilsimulation: ist eine Simulation, die Teil einer Co-Simulation ist. Somit setzt sich einer Co-Simulationen aus mehreren Teilsimulationen zusammen. Komponentensimulationen und Interaktionssimulationen sind Teilsimulationen.

Lebenslauf

Persönliche Daten

28.03.1990	geboren in Backnang

Schulbildung

09/1996 – 07/2000	Grundschule Steinach-Hößlinswart, Berglen
09/2000 – 06/2009	Lessing-Gymnasium-Winnenden, Winnenden, Abschluss Abitur

Wehrdienst/Zivildienst

03/2008 – 12/2012	Freiwillige Feuerwehr Berglen

Studium

10/2009 – 03/2010	Studium des Wirtschaftsingenieurswesen am Karlsruher Institut für Technologie (KIT)
04/2010 – 07/2010	Praktikum bei der Firma Kärcher in Winnenden
10/2010 – 02/2016	Studium der Elektro- und Informationstechnik an der Universität Stuttgart
24.02.2016	Abschluss Master of Science

Berufstätigkeit

03/2016 – 12/2021	Wissenschaftlicher Mitarbeiter am Institut für Automatisierungstechnik und Softwaresysteme der Universität Stuttgart