# Realisierung von selbstorganisierenden Systemen auf Basis von dezentralen Koordinationsprozessen

**Dissertation**

zur Erlangung des akademischen Grades
Dr. rer. nat
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
der Universität Hamburg

**eingereicht beim Fach-Promotionsausschuss Informatik von**
**Thomas Preisler**
**aus Hamburg**

---

**Hamburg, November 2016**

Gutachter:
Prof. Dr. W. Lamersdorf, Universität Hamburg
Prof. Dr. W. Renz, Hochschule für Angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. I. J. Timm, Universität Trier

Tag der Disputation: 30. Januar 2017

*I'm gonna have to science the sh*t out of this.*

Mark Watney (Film: The Martian)

# Zusammenfassung

Das Paradigma der Selbstorganisation hat sich als vielversprechender Lösungsansatz herausgestellt, um moderne, verteilte Anwendungen und Systeme mit hohen Adaptivitätsanforderungen zu realisieren. Dabei hat sich gezeigt, dass dezentrale Steuerungs- und Koordinationskonzepte geeignet sind, um das selbstorganisierenden Verhalten, welches die gewünschte Adaptivität herausbildet, umzusetzen. Die besondere Bedeutung des Paradigmas der Selbstorganisation ergibt sich aus der Möglichkeit, Anwendungen und Systeme zu entwickeln, die zur Laufzeit ein selbstadaptives Verhalten aufweisen. Solche Systeme sind in der Lage, ihre Strukturen und ihr Verhalten an sich verändernde Gegebenheiten anzupassen. Dabei ist es unerheblich, welcher Natur die Ursachen für eine solche Anpassung sind. Diese können z.B. in einer dynamischen Umgebung, wechselndem Benutzerverhalten oder auch Fehlerfälle begründet sein. Anwendungen und Systeme, die solche Eigenschaften aufweisen, sind im Hinblick auf aktuelle Herausforderungen in Informations- und Kommunikationssystemen, wie z.B. die zunehmende Heterogenität, Interaktionsdichte, Dynamik und Komplexität, von besonderer Bedeutung.

Im Rahmen dieser Dissertation erfolgt zunächst eine detaillierte Analyse des Paradigmas der Selbstorganisation, dessen charakteristische Eigenschaften neben der Selbstadaptivität auch erstrebenswerte nichtfunktionale Eigenschaften wie Skalierbarkeit und Robustheit beinhalten. Darauf aufbauend wird die systematische Entwicklung selbstorganisierender Systeme thematisiert, indem die Hauptbestandteile dieser Systeme analysiert und vorgestellt werden. Dabei wird auch gezeigt, dass die softwaretechnische Umsetzung dieses Konzepts eine besondere Herausforderung darstellt, da sich aufgrund des nichtlinearen und nichtdeterministischen Verhalten dieser Systeme unerwünschte emergente Phänomene herausbilden können. Dies führt dazu, dass etablierte Softwareentwicklungsansätze zur Konstruktion selbstorganisierender Systeme nur bedingt geeignet sind, sodass neuartige Entwicklungskonzepte und -ansätze benötigt werden. Daher wird nachfolgend ein bestehender, relativ neuer Ansatz zur Konstruktion von selbstorganisierenden Multiagentensystemen thematisiert. Multiagentensysteme sind eine spezielle Unterklasse von verteilten Systemen, die auf dem Agentenparadigma basieren.

Ausgehend von den vorgestellten Erkenntnissen und Vorarbeiten geht es im Rahmen dieser Dissertation im Speziellen um zwei Fragestellungen: Zunächst wird die Konstruktion eines möglichst allgemeingültigen Rahmenwerks behandelt, das es erlaubt, Anwendungen und Systeme, die auf unterschiedlichen Technologien und Architekturen beruhen, mit selbstadaptivem bzw. selbstorganisierendem Verhalten auszustatten. Die Realisierung dieses Verhaltens erfolgt dabei auf Basis von dezentralen Koordinationsprozessen. Zwar existieren bereits konzeptionelle Vorarbeiten, die sich mit der Konstruktion von selbstorganisierendem Verhalten auf Basis von dezentralen Koordinationsprozessen beschäftigen, diese beschränken sich allerdings auf das Architekturparadigma der Multiagentensysteme. Die unterschiedlichen Komponenten und Sub-Systeme, aus denen sich moderne, verteilte Systemen zusammensetzen, zeichnen sich aber häufig durch eine Heterogenität in Bezug auf die verwendeten Technologien und Architekturen aus. Ein aktuelles Anwendungsgebiet solcher heterogener, verteilter Systeme sind Co-Simulationen, bei denen unterschiedliche, verteilte Simulationskomponenten verwendet werden, um ein komplexe Problem zu formulieren. Um solche Systeme mit selbstorganisierenden Eigenschaften auszustatten, wird ein Koordinationsansatz benötigt, der generische Software-Komponenten im Allgemeinen unterstützt. Da solche Komponenten aber, im Gegensatz zu Agenten, nicht von einer speziellen Ausführungsumgebung verwaltet werden, ergeben sich spezielle Herausforderungen bei der Koordination. Diese werden im Rahmen der ersten Fragestellung herausgearbeitet und ein möglichst allgemeingültiges Rahmenwerk zur Koordination solcher Systeme entwickelt.

Davon ausgehend wird im Rahmen der zweiten Fragestellung die automatisierte Anpassung von Koordinationsprozessen zur Laufzeit thematisiert, um unerwünschte Dynamiken in einem selbstorganisierenden System zu verhindern. Dabei wird gezeigt, wie der Austausch bzw. die Rekonfiguration der Koordinationsprozesse, die die selbstorganisierende Dynamik im System abbilden, dabei hilft, unerwünschtem Verhalten entgegenzuwirken. Solches erwünschtes Verhalten kann sich z.B. in Form einer schlechten Performanz oder der sogenannten „Starvation“ äußern. Bei der Ausführung solcher Anpassungen kann grundsätzlich zwischen zwei Modi unterschieden werden: Parametrische Anpassungen betreffen die dynamische Anpassung von bestimmten Koordinationsparametern, die einen konfigurierenden Einfluss auf die betreffenden Koordinationsprozesse haben. Strukturelle Anpassungen hingegen betreffen die Struktur der Koordinationsprozesse und beziehen sich daher auf einen Austausch der aktiven Prozesse zur Laufzeit.

# Abstract

The paradigm of self-organization has emerged as a promising approach to realize modern, distributed applications and systems with high adaptivity requirements. Decentralized control and coordination concepts have proven their applicability to implement the self-organizing behavior, which creates the desired adaptivity. Of particular importance for the paradigm of self-organization is the possibility to develop applications and systems with self-adaptive run-time behavior. Such systems can adapt their structure and behavior to changing circumstances and conditions. Here, the nature of the adaptation cause is irrelevant, it may be based on e.g. dynamic changes of the environment, changing user behavior or even errors. Applications and systems with such self-organizing characteristics are well-suited for current challenges in information and communication systems, e.g. increasing heterogeneity, interaction density and complexity.

As part of this dissertation, a detailed analysis of the paradigm of self-organization is carried out. In addition to self-adaptivity, the characteristics of self-organizing systems also include desirable non-functional properties such as scalability and robustness. Based on this, the systematic development of self-organizing systems will be addressed by analyzing and presenting the main components of these systems. It is also shown that the implementation of this concept is particular challenging, because of non-linear and non-deterministic behaviors as well as emergent phenomena. These problems lead to the fact that established software development approaches are only conditionally suitable for the construction of self-organizing systems. Hence, novel development concepts and approaches are needed. Therefore, an existing, relatively new approach for the construction of self-organizing multi-agent systems is discussed. Multi-agent systems are a special subclass of distributed systems based on the agent paradigm.

Based on the presented findings and preliminary studies, the thesis deals with two specific questions: First, the design of a framework that is as general as possible and allows to equip applications and systems based on different technologies and architectures with self-adaptive or self-organizing behavior. Thereby, the realization of this behavior is based on decentralized coordination processes. Although conceptual preliminary studies exist that deal with the construction of self-organizing behavior based on decentralized coordination processes, they are, however, limited to the multi-agent systems paradigm. But, modern distributed systems are often characterized by heterogeneous components and sub-systems that form the whole structure. A current application of such heterogeneous, distributed systems are co-simulations where different simulation components are coupled to formulate a complex problem. In order to equip such systems with self-organizing properties, a coordination approach is needed, that supports generic software components in general. However, such components are unlike agents not managed by a special execution platform. Therefore, special challenges arise when it comes to coordinating such components. These are elaborated as part of the first question complex when a coordination framework for generic software components will be developed.

In the context of the second question, the automated adaptation of coordination processes at run-time will be reviewed, in order to prevent unwanted dynamics in self-organizing systems. In this context, it is shown how the exchange or reconfiguration of the underlying coordination processes, responsible for the system's self-organizing dynamic, helps to prevent undesired behavior. Such undesired behavior may occur in form of a performance decrease or even starvation. When performing such adjustments, two modes can be distinguished: Parametric adaptations are used to dynamically adjust certain coordination parameters. These are parameters which configure the corresponding coordination processes. Structural adaptations, on the other hand, concern the overall structure of the coordination processes and therefore refer to an exchange of the active coordination processes at runtime.

# Danksagung

Diese Arbeit ist im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Labor für multimediale Systeme (MMLab) an der Hochschule für Angewandte Wissenschaften (HAW) Hamburg entstanden. Während dieser Zeit haben mich zahlreiche Menschen begleitet und mich und diese Arbeit in positiver Weise beeinflusst. Dafür möchte ich mich herzlich bedanken. Besonderer Dank gilt meinen beiden Doktorvätern Prof. Dr. Winfried Lamersdorf und Prof. Dr. Wolfgang Renz, die meinen wissenschaftlichen Werdegang stets gefördert haben und mir gleichzeitig große Freiheiten gelassen haben, meine eigenen Forschungsinteressen zu entdecken und zu verfolgen. Zusätzlich möchte ich Herrn Renz dafür danken, dass die Finanzierung meiner Stelle fortwährend gewährleistet war und er mir stets mit Rat und Tat zur Seite stand.

Des Weiteren möchte ich mich bei ehemaligen und aktuellen Kollegen an der HAW für eine sehr spannende und schöne Zeit sowie die entstandenen Freundschaften bedanken. Dr. Jan Sudeikat danke ich dafür, dass er maßgeblich mein Interesse an selbstorganisierenden Systemen geweckt hat. Außerdem wären viele wissenschaftliche Publikationen nicht entstanden, wenn Jan mich nicht anfangs unter seine Fittiche genommen hätte. Andreas Ißleib möchte ich für die hervorragende technische Ausstattung meines Arbeitsplatzes und die vielen netten Gespräche in der Kaffeerunde danken. Gregor Balthasar und Kai Rosseburg danke ich für die vielen gemeinsamen Mittags- und Kaffeepausen, die immer eine willkommene Ablenkung waren. Meinem Büronachbarn Tim Dethlefs danke ich für die hervorragende inhaltliche Zusammenarbeit, die vielen gemeinsam entstandenen Publikationen und die Unmenge an kreativen Ideen bzgl. der „Kaffee“pausengestaltung.

Weiterhin möchte ich allen Beteiligten am Arbeitsbereich Verteilte Systeme und Informationssysteme (VSIS) am Fachbereich Informatik der Universität Hamburg danken, die dafür gesorgt haben, dass ich mich dort immer gut aufgenommen fühlte. Dr. Ante Vilenica für die sehr gute Zusammenarbeit innerhalb unseres DFG-Drittmittelprojektes, die den Grundstein für diese Dissertation gelegt hat. Dr. Dirk Bade für die spannenden inhaltlichen Diskussionen und dafür, dass er immer einen Tee für mich übrig hatte, wenn ich zu Besuch war. Christopher Haubeck für die interessanten Gespräche über selbstorganisierende Systeme und (nicht so gut organisierte) Fußballsysteme. Dr. Lars Braubach, Dr. Alexander Pokahr und Dr. Kai Jander für die gute technische Unterstützung und geduldige Beantwortung aller meiner Fragen zur Jadex-Plattform. Anne Hansen-Awizen für die Hilfe bei vielen organisatorischen Fragen und dafür, dass ich nie eine VSIS-Weihnachtsfeier verpasst habe.

Für ihre Zeit und wertvollen Hinweise beim Korrekturlesen bin ich Tim Dethlefs und Eggert Thode zu sehr großem Dank verpflichtet.

Meinen Freunden Eggert Thode, Holger Stief, Jara Frank, Lennart Osterhus und Nadine Schubert möchte ich für ihre Unterstützung und Freundschaft danken und dafür, dass sie mir immer wieder geholfen haben, den Kopf frei zu bekommen und den Blick auf die angenehmen Dinge nicht zu verlieren.

Schließlich möchte ich meiner Familie danken: Meinen Eltern Marit und Ernst und meinem Bruder Andreas für ihre bedingungslose Unterstützung und Ermutigung sowie meinem Onkel Dr. Holger Wille, der mir stets ein Vorbild war.

*Thomas Preisler*

# Inhaltsverzeichnis


**1. Einleitung** **1**

1.1. Gegenstand der Forschung . . . . . 3
1.2. Zielsetzung und Forschungsfragen . . . . . 5
1.3. Ergebnisse und Beiträge . . . . . 7
1.4. Aufbau der Arbeit . . . . . 9

**2. Konzeption und Anwendung selbstorganisierender Systeme** **13**

2.1. Selbstorganisation . . . . . 13
2.1.1. Definition . . . . . 14
2.1.2. Emergenz . . . . . 16
2.1.3. Adaptivität . . . . . 18
2.2. Selbst-Adaptive und Selbst-Verwaltende Systeme . . . . . 19
2.2.1. Dimension adaptiver Systeme . . . . . 20
2.2.2. Hierachie adaptiver Systemeigenschaften . . . . . 21
2.2.3. Autonomic Computing . . . . . 23
2.2.4. Organic Computing . . . . . 25
2.2.5. Unterscheidungsmerkmale adaptiver Systeme . . . . . 27
2.3. Selbstorganisation auf Basis dezentraler Koordination . . . . . 28
2.3.1. Konzeption . . . . . 29
2.3.2. Koordination, Kommunikation und Interaktion . . . . . 30
2.3.3. Koordinationsmechanismen . . . . . 33
2.4. Selbstorganisierende Eigenschaften in verteilten Systemen . . . 37
2.5. Anwendung selbstorganisierender Systeme . . . . . 39
2.5.1. Selbstheilendes Ressourcenflusssystem . . . . . 40
2.5.2. Selbstorganisierendes Fahrradverleihsystem . . . . . 44
2.5.3. Selbstoptimierendes Demand-Side-Management . . . . . 53
2.6. Zusammenfassung . . . . . 58

**3. Systematische Entwicklung selbstorganisierender Systeme** **61**

3.1. Hauptbestandteile selbstorganisierender Systeme . . . . . 61
3.1.1. Feedback-Schleifen . . . . . 62
3.1.2. Koordination . . . . . 64
3.1.3. Agenten und Aktive Komponenten . . . . . 67
3.1.4. Umgebung . . . . . 70
3.2. Entwicklung selbstorganisierender Systeme . . . . . 74
3.2.1. Herausforderungen . . . . . 74
3.2.2. Prinzipielle Entwicklungsstrategien . . . . . 77
3.2.3. Existierende Entwicklungsansätze . . . . . 79
3.3. Zusammenfassung . . . . . 87

**4. Konstruktion selbstorganisierender Multiagentensysteme 89**
4.1. Spezielle Herausforderungen für die Konstruktion . . . . . 89
4.2. Vergleich existierender Ansätze . . . . . 91
4.2.1. Klassifikation . . . . . 91
4.2.2. Bewertung . . . . . 94
4.3. Konstruktionsansatz Coordination Space . . . . . 95
4.3.1. Integrationsarchitektur . . . . . 95
4.3.2. Ausführungsumgebung . . . . . 100
4.3.3. Koordinationsmedien . . . . . 108
4.4. Strukturelle Adaption von Koordinationsprozessen . . . . . 111
4.4.1. Strukturelle Adaption in selbstorganisierenden Multiagentensysteme . . . . . 114
4.4.2. Architekturerweiterung für strukturelle Adaptionen . . . 116
4.4.3. Beschreibung von strukturellen Adaptionen . . . . . 118
4.4.4. Adaptions- und Abstimmungs-Prozess . . . . . 120
4.5. Zusammenfassung . . . . . 123

**5. Skalierbare, verteilte und adaptive Co-Simulationssysteme 125**
5.1. Interoperabilität von Simulationen . . . . . 126
5.1.1. Ansätze zur Interoperabilität von Simulationen . . . . . 127
5.1.2. Servicekomponenten-basierte Architektur für Simulationssysteme . . . . . 128
5.1.3. Skalierbare Integration von 4GL-Modellen . . . . . 130
5.2. Simulation as a Service . . . . . 135
5.2.1. Cloud Computing Technologien . . . . . 135
5.2.2. Existierende Ansätze . . . . . 137
5.2.3. Konzept und Architektur . . . . . 138
5.2.4. Alternativen bei der Implementierung . . . . . 141
5.2.5. Daten-adaptive Simulation als Service . . . . . 145
5.3. Zusammenfassung . . . . . 147

**6. Dezentrale Koordination in verteilten Systemen 151**
6.1. Dezentrale Koordination von Aktiven Komponenten . . . . . 152
6.1.1. Entwicklungsansatz . . . . . 152
6.1.2. Architektur . . . . . 155
6.1.3. Implementierung . . . . . 157
6.1.4. Zusammenfassung . . . . . 159
6.2. DeCoF: Dezentrales Koordinationsframework . . . . . 160
6.2.1. Design und Architektur . . . . . 161
6.2.2. Integrationsansatz . . . . . 164
6.2.3. Beschreibung von Koordinationsprozessen . . . . . 167
6.2.4. Integrationsbeispiel: Jade Party . . . . . 170
6.2.5. Zusammenfassung . . . . . 172
6.3. Strukturelle Adaption von Koordinationsprozessen . . . . . 174
6.3.1. Adaptier- und koordinierbare Komponenten . . . . . 176

6.3.2. Architekturerweiterung für strukturelle Adaptionen . . . 177
6.3.3. Beschreibung von Adaptionen . . . 179
6.3.4. Integrations- und Entwicklungsansatz . . . 182
6.4. Zusammenfassung . . . 184

**7. Anwendung und Bewertung 187**
7.1. Anwendungen . . . 187
7.1.1. MarsWorld Mining Operation . . . 188
7.1.2. Bike-Sharing Washington, D.C. . . . 200
7.2. Bewertung des eigenen Ansatzes . . . 214
7.2.1. Vergleich mit existierenden Ansätzen . . . 214
7.2.2. Bewertung im Hinblick auf die Forschungsfragen . . . 219
7.3. Zusammenfassung . . . 222

**8. Zusammenfassung und Ausblick 225**
8.1. Zusammenfassung . . . 225
8.2. Ausblick . . . 231
8.2.1. Dezentrale Koordination von Co-Simulationen . . . 231
8.2.2. Strukturelle Adaption von Koordinationsprozessen . . . 232
8.2.3. Policen-basierter Ansatz zur Einhaltung nichtfunktionaler Anforderungen . . . 233

**A. Quelltextauszüge 237**

**Eigene Veröffentlichungen 241**

**Abbildungsverzeichnis 245**

**Tabellenverzeichnis 251**

**Quelltextverzeichnis 253**

**Literaturverzeichnis 255**

**Eidesstattliche Versicherung 283**

# 1. Einleitung

Der technologische Fortschritt in der Computer- und Telekommunikationsindustrie in den letzten Jahrzehnten führt zu einer immer weiter zunehmenden Verbreitung und Durchdringung von verteilten Informations- und Datenverarbeitungssystemen (IT). Herrschten vor 30 Jahren noch hauptsächlich Mainframe-Computer vor, die zusammen von mehreren Benutzern verwendet wurden, so ist man heutzutage allgegenwärtig von Computern umgeben. Diese neuen Geräte verfügen i.d.R. über Kommunikationsschnittstellen, die es ihnen z.B. in Form von drahtlosen Funkverbindungen erlauben, miteinander zu kommunizieren und zu interagieren. Diese zunehmende Komplexität von IT-Systemen spiegelt sich aber nicht nur in solchen Einzelgeräten wider. So bestehen Automobile heutzutage aus zahlreichen Prozessoren, Sensoren und eingebetteten Systemen, die zur Steuerung des Fahrzeugs miteinander interagieren. Dabei verhindern sie u.a., dass das Auto von der Straße abkommt oder sorgen dafür, dass im Falle eines Unfalls der Airbag rechtzeitig ausgelöst wird. Gleichzeitig ist das Auto dabei mit der *Cloud* verbunden, um z.B. aktuelle Verkehrsinformationen zu erhalten oder Musik zu streamen. Diese Entwicklung bedingt, dass neue Lebensbereiche von Informations- und Kommunikationstechnologie (IKT) beeinflusst werden, wodurch sich auch für existierende Umfelder neue Möglichkeiten eröffnen. Die Entwicklung ist daher u.a. auch Grund für die zunehmende Bedeutung von Anwendungs- und Forschungsdomänen wie *Mobile Computing* [Rot05], *Ubiquitous Computing* [LY02] und *Ambient Intelligence* [AW09]. Mit dieser Entwicklung geht dabei auch eine Diversifikation von Hard- und Software-Komponenten einher, die neue Herausforderungen für das Management verteilter IKT-Systeme mit sich bringt.

Dabei ergeben sich neben der klassischen Anforderung der *Performanz* auch neue Anforderungen, die auf die *Verfügbarkeit*, *Skalierbarkeit* und *Robustheit* der Systeme hin abzielen [VCO09]. Dies ist nicht zuletzt dadurch bedingt, dass IKT-gestützte Systemen in Bereichen des öffentlichen Lebens oftmals die Nervenzentren industrieller Gesellschaften betreffen. Beispiele hierfür sind die Energieversorgung [DPR15b], Kommunikationsnetzwerke [HL11] oder auch Verkehrsnetze [VRSL09]. In solchen Systemen, die maßgeblich durch untereinander vernetzte Computersysteme gesteuert werden, ist ein dauerhafter und möglichst fehlerfreier Betrieb mindestens genauso wichtig wie eine hohe Performanz. Alle diese unterschiedlichen Arten von IKT-Systemen haben gemeinsam, dass sie aufgrund ihrer Komplexität immer schwieriger zu entwickeln und zu warten sind [KC03]. Es kann davon ausgegangen werden, dass die Komplexität moderner IKT-Systeme die Grenzen der menschlichen Fähigkeit zur Kontrolle in nicht allzu ferner Zukunft übersteigen wird. Auch wenn diese Grenze aktuell noch nicht erreicht wurde, so wird zur Wartung

solcher Systeme bereits heute hochqualifiziertes Fachpersonal benötigt. Dieses spiegelt sich auch in steigenden finanziellen Kosten wider; [SPTU05] spricht in diesem Zusammenhang von einer steigenden finanziellen Belastung pro Computer. Aber auch in anderen Bereichen führt die zunehmende Komplexität von solchen Systemen zu Problemen. Im Automobilsektor z.B. haben Studien gezeigt, dass etwa die Hälfte aller Pannen auf Probleme mit elektrischen Komponenten zurückzuführen sind [BMMSP06].

Während auf der einen Seite die Bedeutung von IKT-Systemen immer weiter zunimmt, führen auf der anderen Seite die wachsende Komplexität, Heterogenität sowie die Interaktion mit dynamischem und schwer vorhersehbaren Umgebungen dazu, dass neben der Relevanz der eigentlichen Funktionalität die Bedeutung nichtfunktionaler Anforderungen zunimmt. Im Zusammenhang mit der zunehmenden Komplexität solcher Systeme spricht man auch von der *Komplexitätskrise* [KC03]. Für die Entwicklung moderner, verteilter IKT-Systeme, die derartige Eigenschaften aufweisen, bieten sich dezentrale Steuerungsansätze an, die im Gegensatz zu ihren zentralen Gegenstücken einen höheren Grad an Fehlertoleranz aufweisen [SR08]. Dezentrale Steuerungsansätze zeichnen sich dadurch aus, dass die einzelnen Systemkomponenten zeitgleich als Anbieter und als Nachfrager von Funktionalität agieren können und die Gesamtsteuerung des Systems verteilt erfolgt. Dadurch skalieren solche Systeme deutlich besser und sind robuster gegenüber Fehlersituationen, die z.B. dadurch entstehen, dass einzelne Systemkomponenten durch Überlastung die Performanz des gesamten Systems senken. Ein besonderer dezentraler Steuerungsansatz ist das Paradigma der Selbstorganisation [HWMH06], das einen vielversprechenden Lösungsansatz bietet, um damit moderne verteilte Systemen zu realisieren, die in der Lage sind, die aufgeführten aktuellen Herausforderungen von IKT-Systemen zu adressieren. Selbstorganisation bezeichnet dabei selbstadaptive Systeme, die primär auf Basis von lokalem Wissen und lokaler Interaktion zwischen den Komponenten in der Lage sind, auf globaler Ebene vordefinierte Systemziele zu erreichen, und das in Abwesenheit einer zentralen Steuerungsinstanz [WH04]. Solche selbstorganisierenden Systeme finden sich besonders in naturwissenschaftlichen Disziplin [Cam06, DMSGK11] wie der Biologie (z.B. Schwärme [CDnJS+03, MMTZ06] oder Zellen [Kar08]), der Physik (z.B. Kristallisation [Hey99, Cam06]) und der Chemie (z.B. die Belousov-Zhabotinsky-Reaktion [Tab06, TT09]) wieder. In diesen und auch anderen Domänen offenbaren selbstorganisierende Systeme unterschiedliche Ausprägungsgrade in Bezug auf Eigenschaften wie Robustheit, Fehlertoleranz, Skalierbarkeit und Adaptivität, die wünschenswerte Eigenschaften für moderne, verteilte IKT-Systeme darstellen.

Grundsätzlich ist Selbstorganisation ein faszinierendes Thema, das nicht nur die Gedanken von Wissenschaftlern und Forschern anregt. So skizzierte der Regisseur James Cameron in seinem Science-Fiction Film „*Terminator*“ aus dem Jahre 1984 eine düstere Zukunftsvision, in der intelligente, selbstorganisierende Maschinen den Großteil der Menschheit versklavt haben. Aber

nicht nur in dieser nicht sehr erfreulichen Zukunftsvision wird das Thema Selbstorganisation aufgegriffen, es gibt zahlreiche weitere Beispiele für Science–Fiction Literatur und Filme, die dieses Thema im Zusammenhang mit intelligenten Maschinen behandeln. Neben der Thematisierung in Literatur und anderen Medien ist das Thema Selbstorganisation auch Gegenstand aktueller Entwicklungen. An der Fachhochschule Gelsenkirchen wird z.B. ein selbstorganisierendes Multiagentensystem verwendet, um die Stundenpläne für die Lehre zu erstellen [Zno06]. In diesem System verhandeln eine Vielzahl von autonomen Agenten in mehreren Runden über die Zusammenstellung des Stundenplans. Dabei werden bereits nach kurzer Rechenzeit gute Ergebnisse generiert, die vorher festgelegte Qualitätsbedingungen erfüllen. Am *Massachusetts Institute of Technology* (MIT) wurden Roboter entwickelt, die sich selbstständig organisieren und völlig autonom Ölteppiche beseitigen. Diese *„Seaswarm“*[1] getauften Maschinen zeigen ein intelligentes Schwarmverhalten und schöpfen mit Hilfe von Nano–Gewebe Öl von der Wasseroberfläche ab. Nach Schätzungen des MIT würden 5000 Seaswarm–Roboter bei einer Ölpest vom Ausmaß des Untergangs der Bohrinsel *„Deep Water Horizont“* im Golf von Mexiko im April 2010 etwa einen Monat für die Beseitigung des Ölteppichs brauchen. Im Vergleich dazu waren dort im Jahr 2010 tatsächlich 800 Schiffe im Einsatz und konnten dabei nur etwa drei Prozent des Öls abschöpfen.

Bei genauerer Betrachtung des Paradigmas der Selbstorganisation wird allerdings deutlich, dass eine Übertragung und Anwendung dieses Paradigmas auf Softwaresysteme traditionelle Entwicklungsmethoden vor neue Herausforderungen stellt. Dieses ist insbesondere dadurch begründet, dass Softwaresysteme i.d.R. dem *Top-down*-Ansatz folgend entwickelt werden. Der Entwurf selbstorganisierender Systeme hingegen folgt überwiegend dem *Bottom-up*-Ansatz (vgl. [ST09, DWSH05]. Als *Top-down* und *Bottom-up* werden zwei entgegengesetzte Wirkrichtungen in Prozessen bezeichnet, die in verschiedenen Zusammenhängen zur Analyse und Synthese verwendet werden. *Top-down* geht dabei vom Abstrakten, Allgemeinen und Übergeordneten schrittweise hin zum Konkreten, Speziellen und Untergeordneten. *Bottom-up* bezeichnet die umgekehrte Richtung. Des Weiteren bedingt die Dynamik von selbstorganisierenden Systemen, dass Simulationen ein wichtiger Bestandteil des Entwicklungsprozesses sind [Edm05b]. Mit Hilfe solcher Systemsimulationen kann überprüft werden, ob die zur Entwurfszeit konzipierten lokalen Interaktionen, die auf mikroskopischer Ebene stattfinden, zur Laufzeit zu den gewünschten Effekten auf makroskopischer Ebene führen.

## 1.1. Gegenstand der Forschung

Diese Arbeit ist Teil einer Forschungsrichtung, die sich damit befasst, wie verteilte Systeme so entworfen und konstruiert werden können, dass sie in der Lage sind, sich adaptiv zu verhalten. Adaptivität stellt inzwischen

[1] `http://senseable.mit.edu/seaswarm/`, aufgerufen am 1. Februar 2017

oft eine zentrale Anforderung an moderne verteilte Systeme dar. Innerhalb dieser Forschungsrichtung gibt es eine Vielzahl an Lösungsansätzen, die sich mit grundsätzlichen oder auch ganz speziellen Aspekten beschäftigen. Ein zentrales Unterscheidungsmerkmal ist hierbei die Art und Weise, wie die Steuerung in solchen Systemen umgesetzt wird. Dies hat unmittelbaren Einfluss darauf, wie das adaptive Verhalten der Systeme erreicht wird. Hier kann insbesondere zwischen zentralen und dezentralen Formen der Kontrolle unterschieden werden. Im Rahmen dieser Arbeit wird dabei letztere betrachtet, da auf Basis eines dezentralen Ansatzes die bereits erwähnten, nichtfunktionalen Anforderungen wie Performanz, Skalierbarkeit, Verfügbarkeit und Robustheit besonders gut umsetzbar sind (vgl. u.a. [SR08, WMA10]). Dabei dient das Paradigma der Selbstorganisation als Ausgangspunkts zur Erreichung von selbstadaptiven Verhalten auf Basis von dezentraler Kontrolle.

Betrachtet man das Paradigma der Selbstorganisation genauer, so zeigt sich, dass dezentrale Koordination von entscheidender Bedeutung ist, um verteilte Systeme mit der Fähigkeit zur Selbstorganisation auszustatten [SR09b, VS09]. Koordination wird an dieser Stelle benötigt, um Abhängigkeiten zwischen autonomen Aktivitäten bzw. Komponenten in einem verteilten System zu verwalten [MC94]. Das grundlegende Potential dezentraler Koordinationsmechanismen zur Umsetzung selbstorganisierenden Verhaltens in natürlichen und technischen Systemen wurde u.a. in [MMTZ06] aufgezeigt. Daraus ergibt sich allerdings die Frage, wie die Vielzahl an Koordinationsmechanismen zielgerichtet für die Entwicklung verteilter Systeme nutzbar gemacht werden kann und welche grundlegende Beziehung zu der eigentlichen Kernfunktionalität eines verteilten Systems besteht. Mit Blick auf [GC92, SBP+09] zeigt sich, dass die softwaretechnische Nutzbarmachung zur systematischen Entwicklung selbstorganisierender Systeme eine grundlegende Trennung oder zumindest eine lose Kopplung zwischen der Kern- und Koordinationsfunktionalität erfordert. Damit ist es möglich, den Aspekt der Selbstorganisation und das damit verbundene selbstadaptive Verhalten sowie die dafür benötigten Koordinationsbemühungen als eine separate Dimension, neben der eigentlichen Kernfunktionalität, bei der Entwicklung verteilter Systeme zu betrachten.

Das Paradigma der Selbstorganisation besitzt also großes Potential zur Entwicklung moderner verteilter Anwendungen und ermöglicht es diesen, aktuelle Herausforderungen, wie die wachsende Heterogenität, Komplexität und Dynamik in IKT-Systemen, zu beherrschen. Gleichzeitig wurde aber bereits deutlich, dass die systematische, softwaretechnische Nutzbarkeit selbstorganisierender Systeme traditionelle Softwareentwicklungsmethoden durch neue Herausforderungen an ihre Grenzen bringt. Zu diesen Herausforderungen gehören u.a. nichtlineares und nichtdeterministisches Verhalten, unvorhersehbare und nicht erwünschte emergente Phänomene sowie Unzulänglichkeiten bei den formalen Methoden zur Analyse des Laufzeitverhaltens. Diese Herausforderungen führen dazu, dass die Mehrheit etablierter Ansätze zur Entwicklung verteilter Anwendungen ungeeignet ist oder für die Entwicklung selbstorganisierender Systeme neu angepasst werden müsste.

Im Rahmen der Dissertationen von *Sudeikat* [Sud10] und *Vilenica* [Vil13] erfolgten wichtige Vorarbeiten im Bezug auf die Entwicklung selbstorganisierender Systeme, insbesondere im Bezug auf die Realisierung der gewünschten selbstorganisierenden Dynamik auf Basis von dezentraler Koordination. Dabei wurde im Rahmen der Dissertation von *Sudeikat* ein Entwicklungsansatz erarbeitet, der die systematische, softwaretechnische Realisierung von selbstorganisierendem Verhalten in verteilten Systemen ermöglicht [SBP+09]. Dazu wurden u.a. eine Beschreibungssprache realisiert [SR09b], die die deklarative Beschreibung von dezentralen Koordinationsprozessen ermöglicht, sodass ein Entwickler die Möglichkeit hat, in deklarativer Form zu beschreiben, *was* im Rahmen des Koordinationsprozesses passieren soll. Ergänzt wurde diese Beschreibungssprache durch die Referenzimplementierung einer Koordinationsschicht [SR09a], die von den so beschriebenen Koordinationsprozessen instruiert wird und die Koordinationsfunktionalität der entsprechenden Anwendung kapselt. Somit ist es möglich, die Koordinationsfunktionalität lose gekoppelt von der eigentlichen Anwendungsfunktionalität zu entwickeln. Im Rahmen der Dissertation von *Vilenica* wurden diese Konzepte im Hinblick auf einem Konstruktionsansatz, der die Koordinationsfunktionalität in einen gesonderten Raum der Umgebung einer verteilten Anwendung auslagert [VSL+10] und Konzepte zur automatisierten Evaluation der Qualität der selbstorganisierenden Dynamik, auf Basis von Simulationen [VL11, VL12], erweitert.

## 1.2. Zielsetzung und Forschungsfragen

Im Rahmen dieser Dissertation geht es im Speziellen um zwei Fragestellungen, welche die Realisierung von modernen, verteilten Systemen auf Basis von dezentralen Koordinationsprozessen betreffen. Zunächst geht es darum, die Vorarbeiten von *Sudeikat* und *Vilenica* dahingehend zu ergänzen, dass ein möglichst allgemeingültiges Rahmenwerk geschaffen wird, welches es erlaubt, Anwendungen, die auf unterschiedlichen Technologien und Konzepten basieren, mit selbstadaptiven bzw. selbstorganisierenden Eigenschaften auf Basis von dezentralen Koordinationsprozessen auszustatten. Die Konstruktion selbstorganisierender Anwendungen basierend auf diesem Konzept beschränkt sich im Kontext der Vorarbeiten auf das Architekturparadigma der Multiagentensysteme [Woo09]. Moderne, verteilte Systeme weisen allerdings häufig eine Heterogenität im Bezug auf die verwendeten Technologien und Architekturen der einzelnen Komponenten bzw. Sub-Systeme auf (z.B. im Umfeld der Co-Simulation [GKL06, RWM+14]). Daher wird ein Koordinationsansatz benötigt, der nicht nur das Agentenparadigma im Speziellen, sondern generische Software-Komponenten im Allgemeinen unterstützt. Generische Software-Komponenten sind im Gegensatz zu Agenten, die innerhalb einer entsprechenden Multiagentenplattform ausgeführt werden, nicht in ihrer Ausführung *verwaltet*. Solche verwalteten Ausführungsumgebungen bieten im Kontext der Multiagentensysteme spezifische Designelemente, aus de-

nen sich das Verhalten der Agenten zusammensetzen lässt (z.B. die *Beliefs*, *Plans* und *Goals* aus dem BDI-Agentenmodell [Bra87]). Zu Koordinationszwecken können diese Designelemente über entsprechende Schnittstellen der Ausführungsumgebung zur Laufzeit observiert und beeinflusst werden, um somit den Zustand der Agenten zu überwachen und ihr Verhalten zu koordinieren. Wird ein Koordinationsansatz für generische Software-Komponenten und heterogene Systeme angestrebt, können solche implizit vorhandenen Designelemente aufgrund der angestrebten Architektur- und Plattformunabhängigkeit nicht verwendet werden, um den Zustand der Komponenten zu observieren und ihr Verhalten zu beeinflussen. Im Rahmen dieser Dissertation werden daher u.a. die folgenden zwei Forschungsfragen behandelt:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen, um koordinierbar zu sein?
2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

Dabei soll eine Lösung gefunden werden, die auf die Verwendung solcher implizit vorhandenen Designelemente verzichtet, um einen generischen Koordinationsansatz zu realisieren. Dieser soll sowohl die Koordination von generischen Software-Komponenten unterstützen als auch eine Lösung bieten, um heterogene Systeme, die auf unterschiedlichen Architekturparadigmen und Ausführungsumgebungen basieren, koordinieren zu können. Ausgehend von diesem Ergebnis gilt es dann im Rahmen der zweiten Forschungsfrage, ein entsprechendes Rahmenwerk zur Koordination zu konstruieren.

Der zweite Fragestellungskomplex beschäftigt sich mit der automatisierten Anpassung von Koordinationsprozessen. Selbstorganisierende Systeme können unter bestimmten Bedingungen ein unerwünschtes Verhalten, z.B. in Form einer Verringerung der Performanz, aufzeigen. Daraus ergibt sich die dritte Forschungsfrage, die im Kontext dieser Arbeit beantwortet wird:

3. Wie kann in einem selbstorganisierenden System dynamischen Ineffizienzen in Form eines unerwünschten Verhaltens entgegen gewirkt werden?

Ein vielversprechendes Konzept zur Überwindung solcher dynamischer Ineffizienzen in selbstorganisierenden Systemen, das im Rahmen dieser Dissertation untersucht wird, ist der dynamische Austausch der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden, bzw. deren Rekonfiguration. Ein solcher Austausch der Koordinationsprozesse wird im Kontext dieser Arbeit als strukturelle Adaption bezeichnet, da dadurch die Struktur des selbstorganisierenden Verhaltens angepasst wird. Im Rahmen dieser Dissertation wird daher untersucht, wie dieses Konzept in dezentral koordinierten Systemen umgesetzt werden kann und wie es dabei konzeptionell in den Entwicklungsprozess selbstorganisierender Systeme eingebunden werden kann.

## 1.3. Ergebnisse und Beiträge

Der zentrale Forschungsaspekt dieser Dissertation beschäftigt sich mit der Realisierung von verteilten Systemen, die ein selbstadaptives oder auch selbstorganisierendes Verhalten aufweisen, welches auf Basis von dezentralen Koordinationsprozessen umgesetzt wird. Als Erweiterung zu den bestehenden Ansätzen und Vorarbeiten wird sich hierbei insbesondere auf die folgenden zwei Aspekte konzentriert: Zunächst geht es um die Konstruktion eines möglichst allgemein gültigen Rahmenwerks, dass es erlaubt, Anwendungen, die auf unterschiedlichen Technologien und Konzepten basieren, mit selbstadaptiven und selbstorganisierenden Eigenschaften auszustatten. Des Weiteren geht es um die strukturelle Anpassung der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden, wenn die selbstorganisierende Dynamik aufgrund von sich verändernden Einflüssen nicht mehr das gewünschte Verhalten erzielt. Zusätzlich wurden im Rahmen dieser Dissertation eine Reihe weiterer Ergebnisse erzielt, die sich mit der Konstruktion von selbstorganisierenden Verhalten auf Basis von dezentraler Koordination, der Validierung dieses Verhaltens mit Hilfe von stochastischen Modellen oder auch der Konstruktion von adaptiven, skalierbaren Simulationssystemen beschäftigen. Im Folgenden werden daher die wichtigsten im Rahmen dieser Dissertation erzielten Ergebnisse und Beiträge anhand ausgewählter Publikationen zusammengefasst:

- Im Rahmen von [PR12] wurde gezeigt, wie die Robustheit eines Ressourcenflusssystems, wie sie in der Produktionslogistik häufig verwendet werden, um Werkstücke zu produzieren, auf Basis eines selbstorganisierenden Ansatzes verbessert werden kann, indem das System mit der Fähigkeit zur Selbstheilung ausgestattet wird. Dadurch wurde das System in die Lage versetzt, im Falle eines Ausfalls einer Komponente im System durch eine geeignete Rekonfiguration, basierend auf einem dezentralen Koordinationsansatz, trotzdem seine ursprüngliche Aufgabe weiter fortzusetzen. Dabei wurde auch gezeigt, dass der dezentrale Ansatz mit wachsender Systemgröße skaliert.

- Aufbauend auf dem Vorherigen wurde in [RPS12] aufgezeigt, wie mesoskopische stochastische Modelle, die eine Kombination der mikroskopischen und makroskopischen Eigenschaften des Systems abbilden, verwendet werden können um das selbstorganisierende Verhalten eines Systems, in diesem Fall die Selbstheilung des Ressourcenflusssystems, zu validieren.

- Im Rahmen von [PRV13, PDR15a] wurde ein selbstorganisierender Ansatz konstruiert, der auf Basis von dezentralen Koordinationsprozessen die Verfügbarkeit von Fahrrädern an den Ausleihstationen in einem Fahrradverleihsystem erhöht, indem dazu den Benutzern des System alternative Stationen zur Ausleihe und Rückgabe von Fahrrädern vorgeschlagen werden.

- In [PVR13] wurde ein Ansatz entwickelt, der es erlaubt, das im Rahmen der Dissertation von Vilenica [Vil13] entwickelte Konzept des *Coordination Space* [VSL+10] auch verteilt über mehrere Netzwerkknoten zu verwenden. Dazu wurde ein Peer-to-Peer Ansatz verwendet, der den *Coordination Space* über mehrere Netzwerkknoten hinweg verteilt.

- Im Rahmen von [PBDR14a, PBDR14b, PDR15c] wurden Konzepte entwickelt um auf Basis einer Servicekomponenten-Architektur skalierbare und adaptive Co-Simulationssysteme zu entwickeln. Dabei wurde sich insbesondere mit dem Aspekt der Bereitstellung von Simulationsfunktionaltät als Dienst im Rahmen einer *Cloud Computing* Infrastruktur befasst.

- Eine Untersuchung, inwiefern die Steuerung der Verbraucherseite im Rahmen des *Smart Grids* von selbstadaptiven Eigenschaften im Bezug auf eine Selbstoptimierung profitieren kann, erfolgte in [DPR15a]. Hier wurden Konzepte wie *Ant Colony Optimization* und Feedback-Schleifen miteinander verbunden, um ein adaptives Optimierungssystem zur Steuerung des Energieverbrauchs im *Smart Grid* zu realisieren.

- Das Konzept der strukturellen Adaption von Koordinationsprozesse, die das selbstorganisierende Verhalten in einem verteilten System abbilden, wurde in [PR15, PRD15, PDR16c] präsentiert. Dabei geht es darum, auf sich verändernde Einflüsse zu reagieren, die dafür sorgen, dass die selbstorganisierende Dynamik nicht mehr das gewünschte Verhalten zeigt.

- Ein konzeptioneller Zwischenschritt von der Koordination von Multiagentensystemen hin zu einer Unterstützung von generischen Software-Komponenten wurde in [PDR15b] präsentiert. Hier wurde eine Middleware[2] entworfen, die es erlaubt, Anwendungen zu koordinieren, die auf Basis des *Aktiven Komponenten* Ansatzes [PB13] realisiert sind.

- Die Konstruktion eines möglichst allgemein gültigen Rahmenwerks, das es erlaubt, Anwendungen, die auf unterschiedlichen Technologien und Konzepten basieren, mit selbstadaptiven und selbstorganisierenden Eigenschaften auszustatten, wurde schließlich in [PDR16a, PDR16b] präsentiert. Hier wurde ein dezentrales Koordinationsrahmenwerk (DeCoF[3]) vorgestellt, das einen Ansatz liefert um generische Software-Komponenten mit der Fähigkeit zur Koordination auszustatten.

[2] Englisch für Dienstschicht oder Zwischenanwendung. Eine Middleware bezeichnet in der Informatik eine anwendungsneutrale Schicht, die so zwischen Anwendungen vermittelt, dass die Komplexität dieser Anwendungen und ihre Infrastruktur verborgen werden. Eine Middleware kann auch als Verteilungsplattform verstanden werden, welche die Kommunikation zwischen Prozessen unterstützt.

[3] Englisch: „Decentralized Coordination Framework“

## 1.4. Aufbau der Arbeit

Ausgehend von der dargestellten Motivation, dem Gegenstand der Forschung und der daraus resultierenden Zielsetzung sowie den Forschungsfragen ergibt sich folgender Aufbau dieser Arbeit:
Zunächst wird im 2. Kapitel auf die Konzeption und Anwendung selbstorganisierender Systeme eingegangen. Dabei wird zunächst das Paradigma der Selbstorganisation detailliert vorgestellt und analysiert. Das Ziel hierbei ist es, aufzuzeigen, dass das Konzept der Selbstorganisation geeignet ist, um moderne verteile Systeme zu realisieren, die in der Lage sind, die eingangs erwähnten Herausforderungen, denen diese Systeme ausgesetzt sind, zu bewältigen. Dabei wird deutlich gemacht, welche Eigenschaften Selbstorganisation auszeichnet und warum diese gerade für verteilte Systeme von Interesse sind. Dazu werden mit Hilfe von Klassifizierungsansätzen unterschiedliche Ausprägungen von Selbstadaptivität in verteilten Systemen aufgezeigt. Hierbei findet insbesondere eine Abgrenzung zwischen Ansätzen, die auf zentraler bzw. dezentraler Steuerung basieren, statt. Für selbstorganisierende Systeme, die zur Klasse der dezentralen Ansätze gehören, wird gezeigt, dass Koordination dabei ein integraler Bestandteil ist. Hierzu wird auf konzeptioneller Ebene ein Modell vorgestellt, das die Beziehung zwischen der Adaptionsdynamik und der Koordinationsfunktionalität in verteilten Systemen herstellt. Indem charakteristische Eigenschaften der Selbstorganisation betrachtet und in Bezug zu nichtfunktionalen Eigenschaften verteilter Systeme gesetzt werden, wird die Bedeutung des Paradigmas Selbstorganisation für solche Systeme weiterführend analysiert werden. Schließlich werden Beispiele für verteilte Anwendungen aus unterschiedliche Anwendungsdomänen präsentiert, die jeweils über unterschiedliche selbstorganisierende Eigenschaften verfügen und von diesen profitieren.

Das 3. Kapitel befasst sich im Anschluss mit der Fragestellung, wie die systematische Entwicklung verteilter Systeme, die mit selbstorganisierenden Eigenschaften ausgestattet werden sollen, unterstützt werden kann. Hierzu werden zunächst die Hauptbestandteile selbstorganisierender Systeme identifiziert und analysiert. Dabei wird u.a. deutlich, dass Feedback-Schleifen von essentieller Bedeutung sind, um die Dynamik in solchen Systemen zielgerichtet zu erwirken. Außerdem wird beschrieben, welche Bedeutung Softwareagenten und ihre ggf. explizit modellierte Umgebung im Kontext der Konstruktion selbstorganisierender Systeme haben. Ferner werden typische Herausforderungen thematisiert, die sich durch die Nutzung des Paradigmas der Selbstorganisation in verteilten Systeme ergeben. Darauf Bezug nehmend werden existierende Ansätze untersucht und bewertet, die sich die Unterstützung der systematischen Entwicklung selbstorganisierende Systeme zum Ziel gemacht haben.

Im 4. Kapitel wird zunächst auf die speziellen Herausforderungen eingegangen, die sich bei der Konstruktion selbstorganisierender Systeme ergeben. Darauf aufbauend werden eine Reihe von existierenden Konstruktionsansätzen

für selbstorganisierende Systeme klassifiziert und bewertet. Schließlich wird das Konzept des *Coordination Space* zur systematischen Konstruktion selbstorganisierender Multiagentensysteme vorgestellt. Dieses Konzept beinhaltet auf konzeptioneller Ebene eine Integrationsarchitektur, die die transparente und nichtinvasive Integration dezentraler Koordinationsfunktionalität in Softwareagenten anleitet. Zudem wird eine Ausführungsumgebung vorgestellt, die eine Referenzimplementierung des *Coordination Space* Konzepts darstellt. Anschließend werden exemplarisch verteilte Koordinationsmedien vorgestellt, die auf unterschiedliche Art und Weise die dezentrale Koordination von selbstorganisierenden Multiagentensystemen über die Grenzen von Plattformen und Netzwerkknoten ermöglichen. Aufbauend auf diesen Konzepten und Realisierungen wird in diesem Kapitel dann auf die strukturelle Adaption von Koordinationsprozessen eingegangen. Dabei wird eine Lösung aufgezeigt, wie einem unerwünschtes Verhalten in einem selbstorganisierenden System, das unter bestimmten Bedingungen auftritt, entgegengewirkt werden kann, indem es zu einem dynamischen Austausch bzw. der Rekonfiguration der Koordinationsprozesse kommt, die das selbstorganisierende Verhalten der Systems abbilden.

Das 5. Kapitel gibt einen Überblick über skalierbare, verteilte und adaptive Simulationssysteme. Simulationen spielen nicht nur eine entscheidende Rolle bei der Validierung und Evaluation von selbstorganisierenden Systemen, als Teil eines iterativen Entwicklungsprozesses, sondern profitieren ihrerseits auch von selbstorganisierenden Verhalten oder Koordination. Von besonderem Interesse sind in diesem Zusammenhang Co-Simulationen. Bei einer Co-Simulation formen unterschiedliche Sub-Systeme zusammen ein gekoppeltes Problem. Sowohl die Modellierung als auch die Ausführung der Simulation erfolgt dabei verteilt. Unter der verteilten Modellierung wird in diesem Zusammenhang die Verwendung von unterschiedlichen Werkzeugen, Programmen oder Programmiersprachen verstanden. Während der Ausführung tauschen die unterschiedlichen Simulationssysteme dabei Daten aus. Die Interaktion erfolgt dabei i.d.R. über im Vorfeld definierte Schnittstellen. Die unterschiedlichen Sub-Systeme bzw. -Simulationen erscheinen untereinander dabei jeweils als *„Black-Box“*[4]. Die sinnvolle Orchestration bzw. Koordination solcher Co-Simulationssysteme, die aus unterschiedlichen Sub-Simulationen bestehen, ist dabei eine der wichtigsten Herausforderungen. Im Rahmen dieses Kapitels wird ein Überblick darüber gegeben, warum Co-Simulationen bedeutsam für die Entwicklung und Planung zukünftiger *Smart Cities* sind und welche speziellen Herausforderungen sich dabei bei ihrer Konstruktion ergeben.

Das 6. Kapitel stellt einen neuartigen Ansatz zur Realisierung von dezentraler Koordination in verteilten Systemen vor. Der im Rahmen dieser Arbeit entstandene Ansatz greift dabei einige Konzepte aus der Integrationsarchitek-

[4] Als Black Box bezeichnet man in der Systemtheorie und Informatik ein möglicherweise komplexes System, von dem nur das äußere Verhalten betrachtet werden soll. Die interne Struktur bleibt dabei verborgen und die Eigenschaften des Systems können nur anhand der Eingaben und Ausgaben betrachtet werden.

tur des *Coordination Space* Konzepts bzw. dessen Vorarbeiten auf, die speziell die Konstruktion selbstorganisierender Multiagentensysteme im Fokus hatten, und adaptiert sie für ein allgemeineres Konzept, das die Konstruktion selbstorganisierender Systeme unabhängig von der verwendeten Systemarchitektur erlaubt. Durch den von der Systemarchitektur unabhängigen Ansatz wird u.a. die Koordination von Co-Simulationssystemen ermöglicht. Damit beschäftigt sich dieses Kapitel direkt mit den Forschungsfragen aus dem ersten Fragenkomplex, indem beschrieben wird, wie die Koordination generischer Software-Komponenten erfolgen kann, die nicht von einer Ausführungsumgebung verwaltet werden und somit über keine impliziten Designelemente verfügen, welche zu Koordinationszwecken verwendet werden können. Im Rahmen dieses Kapitels werden daher die Eigenschaften identifiziert, über die solche Komponenten verfügen müssen, damit ihr Verhalten zu Koordinationszwecken observiert und beeinflusst werden kann. Daraus resultierend erfolgt die Beantwortung der zweiten Forschungsfrage, in dem ein Rahmenwerk präsentiert wird, das auf Basis dieser identifizierten Eigenschaften die Koordination generischer Software-Komponenten ermöglicht.

Im 7. Kapitel wird zunächst die Durchführung zweier unterschiedlicher Anwendungsstudien beschrieben. Mit Hilfe dieser Anwendungsstudien werden dabei das Konzept zur strukturellen Adaption von Koordinationsprozessen sowie das entwickelte dezentrale Koordinationsframework im Bezug auf ihre Einsatzfähigkeit validiert. Anhand der ersten Fallstudie wird dazu die Anwendbarkeit des Konzepts der strukturellen Adaption von Koordinationsprozessen beschrieben und untersucht. Dabei werden die Koordinationsprozesse zwischen autonomen Robotern, die auf dem Mars Erz abbauen, automatisch adaptiert, sobald die vorgesehene selbstorganisierende Dynamik aufgrund von externen Umwelteinflüssen nicht mehr erreicht werden kann. In der zweiten Fallstudie wird anhand eines selbstorganisierenden Fahrradverleihsystems die Verwendung des in Kapitel 6 präsentierten dezentralen Koordinationsrahmenwerks aufgezeigt. Dabei wird insbesondere darauf eingegangen, wie sich dezentrale Koordinationsprozesse mit Hilfe einer deklarativen Beschreibungssprache modellieren lassen und welche technischen Integrationsschritte aus Sicht eines Entwicklers nötig sind, um generische Softwarekomponenten mit der Fähigkeit zur Koordination auszustatten. Ergänzt wird dieses Kapitel durch eine Bewertung der in dieser Dissertation erzielten Ergebnisse. Dazu werden existierende Ansätze mit einer ähnlichen Zielsetzung vorgestellt und insbesondere mit dem entwickelten dezentralen Koordinationsrahmenwerk verglichen. Schließlich erfolgt eine abschließende Bewertung der Ergebnisse im Bezug auf die formulierten Forschungsfragen.

Eine zusammenfassende Betrachtung dieser Arbeit findet schließlich im 8. Kapitel statt. In diesem Kapitel werden die erzielten Ergebnisse zusammengefasst und es wird ein Ausblick auf mögliche zukünftige Arbeiten in diesem Forschungsgebiet gegeben.

# 2. Konzeption und Anwendung selbstorganisierender Systeme

Aufgrund der zunehmenden Komplexität und Vernetzung aktueller verteilter Systeme sind neuartige Konzepte für deren Entwicklung und Betrieb erforderlich geworden. Im Kern hat diese Entwicklung zur Folge, das moderne verteilte Systeme in der Lage sein müssen, sich zur Laufzeit automatisch an neue und zum Teil unbekannte Situationen anzupassen. Dabei gilt es, nichtfunktionale Eigenschaften wie z.B. Robustheit und Skalierbarkeit zu gewährleisten. Ein vielversprechender Ansatz ist hierbei das Konzept der *Selbstorganisation*, das seinen Ursprung in natürlichen Systemen hat und über Eigenschaften verfügt, von denen auch moderne, verteilte Systeme profitieren können, um die eingangs erwähnten Herausforderungen zu meistern. In diesem Kapitel wird daher das Konzept der Selbstorganisation genauer beschrieben. Es wird deutlich gemacht, was Selbstorganisation auszeichnet und welche Paradigmen dabei für verteilte Systeme von Interesse sind.

## 2.1. Selbstorganisation

Das Konzept der Selbstorganisation hat seine Ursprünge in der Studie natürlicher Systeme [Hak06]. Hierbei gibt es eine Vielzahl von Beschreibungen und Definitionen. Eine historische Übersicht über das Konzept der Selbstorganisation findet sich in der Dissertation von *Shalizi* [Sha01]. In verschiedenen Arbeiten (z.B. [Pro13]) wird die erstmalige Erwähnung des Begriffs *Selbstorganisation* in den wissenschaftlichen Publikationen des letzten Jahrhunderts *William Ross Ashby* [Ash47] zugesprochen. Das Konzept der Selbstorganisation beschreibt das Auftreten von kollektiven Phänomenen innerhalb natürlicher, z.B. physikalischer, biologischer oder sozialer Systeme. Innerhalb dieser Systemmodelle formieren sich Organisationsstrukturen ohne kontrollierenden Einfluss von außen durch lokale Interaktionen zwischen den autonomen Entitäten (z.B. Partikel, Zellen, usw.). Viele Beispiele für selbstorganisierende Systeme lassen sich in der Biologie antreffen. Eines der prominentesten Beispiele für ein solches biologisches, selbstorganisierendes System sind die Pheromonspuren, die Ameisen bei der Futtersuche auslegen [LPK+14]. Einzelne Ameisen wandern bei der Futtersuche zunächst scheinbar chaotisch umher. Sobald eine Ameise allerdings eine Futterquelle entdeckt, bringt sie einen Teil davon zum Nest zurück und hinterlässt dabei eine Spur aus Pheromonen. Andere Ameisen entdecken diese Spuren bei ihrer Erkundung des Gebiets und folgen ihnen. Da zunächst nur wenige Pheromone den Weg zur Futterquelle weisen, ist das Verhalten der Ameisen zu Beginn noch recht unkoordiniert. Da es zahlreiche Ameisen sind, gehen sie viele unterschiedliche

Wege zur Futterquelle und zurück zum Nest. Dabei hinterlassen sie immer neue Pheromonspuren. Da Pheromone flüchtig sind, ist die Geruchsspur umso stärker, je kürzer der Weg ist. Dadurch wählen immer mehr Ameisen den kürzesten Weg und hinterlassen dabei wiederum ihrerseits Pheromone, was in einem selbstverstärkenden Effekt resultiert. Darüber hinaus findet sich das Konzept der Selbstorganisation auch in weiteren, unterschiedlichen Bereichen der Wissenschaft wieder. Im Folgenden sind exemplarisch einige davon aufgelistet:

- Schwarmverhalten in der Biologie [MMTZ06]
- Kristallbildungen in der Physik [Hey99]
- Belousov-Zhabotinsky-Reaktion in der Chemie [TT09]

Natürlich gibt es auch innerhalb der Informatik viele Teilgebiete und Anwendungsdomänen, in denen das Konzept der Selbstorganisation eingesetzt wird. Anwendung findet Selbstorganisation so z.B. in der Klassifizierung von Daten im Bereich des *Data-Mining* [Wol07], bei der Koordination modularer Roboter [MMTZ06] und autonomer Fahrzeuge [Wol07], im Bereich der Sensor- und Peer-to-Peer-Netzwerke [PB05] sowie bei der Produktions- [JB03] und Verkehrssteuerung [Ger07]. Aktuell finden Konzepte der Selbstorganisation auch Anwendung im Bereich der intelligenten Stromnetze [DPR15a].

### 2.1.1. Definition

Insgesamt wird deutlich, dass es sich bei Selbstorganisation um ein interdisziplinäres Forschungsgebiet handelt. Dementsprechend finden sich zahlreiche Definitionen in der Literatur, die jeweils spezielle Aspekte hervorheben. Eine Übersicht über diese unterschiedlichen Definitionen wird in [And02] gegeben. Im Folgenden werden, basierend auf [Pro13], drei etablierte Definitionen aufgegriffen, um die wesentlichen Charakteristika von selbstorganisierenden Systemen zu verdeutlichen. Den Anfang macht dabei die Definition von *Camazine et. al.:*

> *Self-organization is a process in which pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system.*
>
> [CDnJS+03, Seite 8]

Hier wird betont, dass es bei einem selbstorganisierenden Prozess zwei wichtige Ebenen gibt: Die globale makroskopische Ebene und die darunterliegende mikroskopische Ebene. Auf der mikroskopischen Ebene werden dabei die Konfiguration, das Verhalten und die Interaktionen der einzelnen Komponenten betrachtet. Demgegenüber betrachtet die makroskopische Ebene das hieraus resultierende dynamische Verhalten aus einer globalen Perspektive heraus und abstrahiert dabei von den einzelnen Komponenten. Bezieht

man diese Betrachtungsweise auf die Modellierung von verteilten, selbstorganisierenden Systemen, so resultiert es typischerweise in einem *Bottom-up*-Entwicklungsprozess, der im Gegensatz zu klassischen *Top-down*-Ansätzen steht.

Ergänzend dazu passt die Definition von *Haken*, die nochmal explizit hervorhebt, dass in einem selbstorganisierenden System Strukturen aus sich selbst heraus gebildet werden und dies ohne spezifischen Eingriff von außen geschieht:

> *A system is self-organizing if it acquires a spatial, temporal or functional structure without specific interference from the outside.*
>
> [Hak06, Seite 11]

Die Definition von *Cools et al.* betont schließlich sowohl die Interaktionen zwischen den einzelnen Elementen in einem selbstorganisierenden System als auch das daraus resultierende dynamische Erzielen eines globalen Verhaltens. Anhand dieser Definition wird also erneut herausgestellt, dass ein selbstorganisierendes System durch Interaktionen zwischen den einzelnen Elementen ein system-globales Verhalten erreicht.

> *A system described as self-organizing is one in which elements interact in order to achieve dynamically a global function or behavior.*
>
> [CGD13, Seite 42]

Im Rahmen dieser Arbeit wird die Definition, die in der der Dissertation von *Vilenica* [Vil13] erarbeitet wurde, verwendet, da diese die wichtigsten Kernpunkte der vorherigen Definitionen kombiniert und die wesentlichen Eigenschaften, die ein selbstorganisierendes System besitzen muss, betont:

> *Selbstorganisation bezeichnet einen dynamischen und adaptiven Prozess. Dieser basiert auf lokalen Interaktionen auf mikroskopischer Ebene und ermöglicht es Systemen aus sich selber heraus, also in Abwesenheit einer externen Kontrollinstanz, Strukturen auf makroskopischer Ebene herauszubilden, beizubehalten und an neue Situationen anzupassen.*
>
> [Vil13, Seite 13]

Die Definition von *Vilenica* liegt damit u.a. im Einklang mit [HWM06], wo eine Methodologie für die Klassifizierung selbstorganisierender Systeme erarbeitet wurde. Mit dem dort präsentierten Rahmenwerk werden Forscher in die Lage versetzt, ihre Systeme zu klassifizieren, um Aussagen darüber treffen zu können, *warum* und in *welcher Art* ihre Systeme selbstorganisierend sind.

Eine weitere wichtige Eigenschaft von selbstorganisierenden Systemen ist die inhärente Dezentralität der Kontrolle über die Strukturformation (vgl. u.a. [Hey99, BC06, SGK06]). Die Kontrolle über die Formation der Struktur wird dabei von allen Systemelementen geteilt. Wenn nur eine Untermenge der Systemelemente für die Formation der Struktur verantwortlich wäre, würde deren hypothetische Entfernung die Fähigkeit des Systems zur Selbststrukturierung eliminieren. Das System wäre demnach entsprechend der Definition nicht mehr selbstorganisierend.

### 2.1.2. Emergenz

Im Kontext von selbstorganisierenden Systemen wird häufig auch von *Emergenz* gesprochen. Emergenz bezeichnet dabei die Herausbildung von neuen Eigenschaften oder Strukturen eines Systems, hervorgerufen durch das Zusammenspiel seiner Elemente. Im Zusammenhang mit Emergenz spricht man häufig davon, dass das Ganze mehr ist als die Summe seiner Einzelteile. Ähnlich wie Selbstorganisation ist Emergenz ein interdisziplinäres Forschungsfeld. Emergente Phänomene werden z.B. in der Physik, Chemie, Biologie, Psychologie oder Soziologie beschrieben. So wird z.B. in der Philosophie des Geistes [Kim95], einem Teilbereich der Philosophie, von einigen Philosophen die Meinung vertreten, dass das Bewusstsein eine emergente Eigenschaft des Gehirns sei. Analog zur Herausbildung von neuen Eigenschaften spricht bei man deren Eliminierung von *Submergenz*. Eine Übersicht über verschiedene Systeme und Modelle, die Emergenz aufweisen, findet sich in der Arbeit von *Holland* [Hol99] wieder.

Der Unterschied zwischen Selbstorganisation und Emergenz wird in der Arbeit von *De Wolf* und *Holvoet* diskutiert [WH04]. Im Rahmen dieser Arbeit wurde u.a. eine Definition von Emergenz aufgestellt:

> *A system exhibits emergence when there are coherent emergents at the macro-level that dynamically arise from the interactions between the parts at the micro-level. Such emergents are novel w.r.t. the individual parts of the system.*
>
> [WH04, Seite 3]

Die Definition verdeutlicht, dass es sich bei Emergenz, ähnlich wie bei Selbstorganisation, um einen dynamischen Prozess handelt, der auf Interaktionen beruht. Eine weitere Gemeinsamkeit besteht in Strukturen, die auf einer makroskopischen Ebene durch die Interaktionen von Elementen auf einer mikroskopischen Ebene entstehen. Darüber hinaus stellt die Definition heraus, dass die Neubildungen (*emergents*) makroskopische Systementitäten sind, die aus dem Zusammenwirken der mikroskopischen Entitäten entstehen. Das Novum an dieser Definition ist, dass es keine explizite Repräsentation der emergenten Neubildungen gibt, sondern diese sich aus dem dynamischen Zusammenwirken der mikroskopischen Entitäten herausbilden. Emergente Phänomene können nach Ansicht von *van Dyke Parunak* und *Brueckner* [VDPB04] als Unterkategorie von selbstorganisierenden Phänomenen verstanden werden.

In [WH04] wird der Zusammenhang zwischen Emergenz und Selbstorganisation anhand von Abbildung 2.1 erklärt. Hierbei wird zunächst deutlich, das Selbstorganisation und Emergenz sowohl zusammen als auch getrennt auftreten können. Abbildung 2.1(a) illustriert den Aspekt, dass in einem System nur Selbstorganisation auftritt. Dies wird durch den Pfeil verdeutlicht, der die interne Organisation symbolisiert. Dadurch, dass es keinen Mikro-Makro-Effekt (implizite dynamische Herausbildung von makroskopischen Strukturen aufgrund von Interaktionen auf mikroskopischer Ebene) gibt, tritt in diesen

Systemen keine Emergenz auf. Ein Beispiel für eine Klasse solcher Systeme sind klassische Multi-Agenten Systeme [VDPB04] in denen die einzelnen Agenten autonom handeln, aber gleichzeitig ein Modell von dem globalen Ziel besitzen, welches sie erreichen wollen, weswegen sie ihr Handeln entsprechend ausrichten. Daher ist die auf globaler Ebene entstehende Struktur keine neue Systemeigenschaft. Abbildung 2.1(b) zeigt ein System, welches nur ein emergentes Verhalten aufweist. Hier weist das System einen Mikro-Makro-Effekt auf, es fehlen aber die typischen Eigenschaften, die ein selbstorganisierendes System auszeichnen. Emergenz ohne Selbstorganisation kann man z.B. in bestimmten thermodynamischen Prozessen [Sha01] beobachten. Betrachtet man z.B. Gas, das ein bestimmtes Volumen hat, dann ist dieses Volumen eine emergente Eigenschaft, die aus den Interaktionen zwischen den einzelnen Partikeln resultiert. Abbildung 2.1(c) zeigt schließlich ein System, dass beide Eigenschaften (Selbstorganisation und Emergenz) aufweist. Ein solcher Fall kann ebenfalls u.a. in Multiagentensystemen auftreten [SGK06]. In diesem Fall besitzen die Agenten nur lokale Strategien und kein explizites Modell der übergeordneten makroskopischen Ebene.

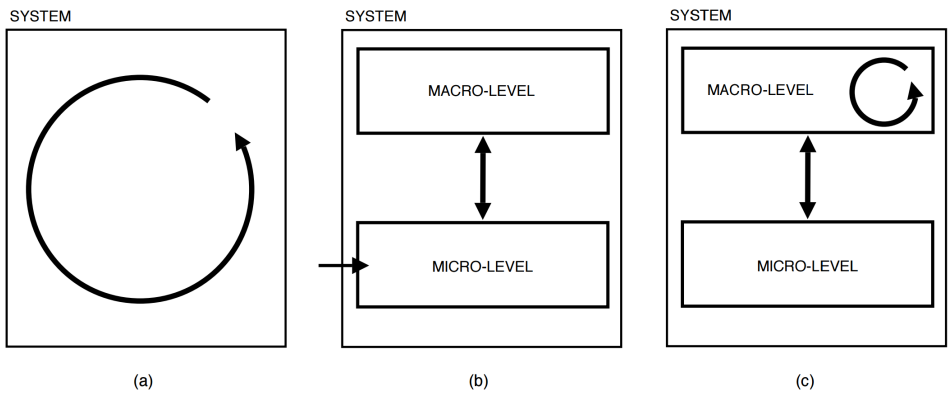



**Abbildung 2.1.:** Zusammenhang zwischen Emergenz und Selbstorganisation: (a) Selbstorganisation ohne Emergenz. (b) Emergenz ohne Selbstorganisation. (c) Emergenz und Selbstorganisation - aus (WH04, S. 10).

Diese Dissertation beschäftigt sich mit der Konstruktion von selbstorganisierenden Verhalten auf Basis von dezentralen Koordinationsprozessen (siehe Abschnitt 2.3). Die Mittel für den Aufbau solcher Systeme sind die Koordinationsprozesse, welche die entsprechenden Strukturen entstehen lassen. Diese Strukturen können verwendet werden, um das gewünschte Verhalten auf der Systemebene zu steuern oder um emergente Phänomene zu kontrollieren. Im weiteren Verlauf dieser Arbeit wird daher zwischen Selbstorganisation und Emergenz nicht weiter unterschieden. Stattdessen liegt der Fokus auf der Strukturbildung selbst und wie diese modelliert, konstruiert und in Softwaresysteme integriert werden kann.

### 2.1.3. Adaptivität

Damit verteilte Systeme in heterogenen, dynamischen und schwer vorhersehbaren Kontexten ihre funktionalen und nichtfunktionalen Anforderungen erfüllen können[1], müssen sie in der Lage sein, sich auf neue Situationen entsprechend einstellen zu können. Eine grundlegende Eigenschaft hierbei ist *Flexibilität*, d.h. in diesem Kontext die Fähigkeit, die Systemstruktur ändern zu können. Wird diese Grundvoraussetzung dazu genutzt, um ein System auf neue Umgebungsanforderungen anzupassen, so spricht man in diesem Kontext von *Adaptivität*. Dieser Begriff wurde in der Informatik schon früh von *Zadeh* geprägt und basierend auf einer mathematischen Sichtweise definiert [Zad63]. Das Ziel der aufgestellten Definition ist es, die Eigenschaft zu quantifizieren, wie gut ein System darin ist, seine Leistungsfähigkeit in verschiedenen, veränderlichen Umgebungssituationen zu erbringen. Die gegebene Aktualität und Relevanz der Definition von *Zadeh* wird u.a. durch *Reinecke et al.* [RWvM10] unterstrichen. Dort wird Adaptivität als die Fähigkeit eines Systems, sich auf eine oder mehrere Leistungsmetriken hin anzupassen, beschrieben. Generell stellt sich bei der Betrachtung adaptiver Systeme die Frage, woher die Fähigkeit zur Umsetzung einer Anpassung kommt. In der Dissertation von *Vilenica* [Vil13] werden dabei das englische Verb „*to adapt*“ und seine Interpretationen betrachtet. Dabei wird zwischen einer transitiven und intransitiven Interpretation unterschieden. Erstere wird im Englischen als „*to make fit*“ interpretiert und bezeichnet den Fall, dass adaptive Systeme von außen, d.h. durch eine dritte Partei, angepasst werden. Die intransitive Form wird im Englischen hingegen mit „*to become adapted*“ interpretiert. In diesem Fall geschieht die Anpassung aus dem System heraus selber. Dementsprechend ist im Englischen im transitiven Fall von „*adaptability*“ und im intransitiven Fall von „*adaptivity*“ die Rede. Im Deutschen, wo eine solche Unterscheidung sprachlich schwierig ist, wird deswegen häufig auch von *Selbstadaptivität* gesprochen, wenn der intransitive Fall gemeint ist. Im Rahmen dieser Arbeit werden verteilte Systeme betrachtet, welche die Fähigkeit besitzen, aus sich heraus, d.h. ohne externen Einfluss, auf neue Anforderungen zu reagieren und sich anzupassen. Daher wird der Begriff der Adaptivität im Folgenden generell aus transitiver Sicht verwendet. Um diese Perspektive zu betonen, kann auch der Begriff Selbstadaptivität verwendet werden.

Weitere interessante Eigenschaften im Bezug auf Adaptivität sind, *wann* und *wie* diese herbeigeführt wird. Bezüglich des erstes Aspekts wird in [Gei08] zwischen *Entwurfszeit-* und *Laufzeitadaptivität* unterschieden. Die erste Form beschreibt dabei den Fall, dass Änderungen an der Softwarearchitektur einer Anwendung offline durchgeführt werden, z.B. weil sich Anforderungen geändert haben oder neue Funktionen hinzugefügt werden sollen (vgl. [BH06, ESW07]). Dieser Ansatz wird auch als statische Adaption bezeichnet und im weiteren Verlauf nicht weiter untersucht. Im Gegensatz dazu beschreibt Laufzeitadaptivität den weitaus anspruchsvolleren Fall, bei dem

[1] Wird im Folgenden als *Leistungsfähigkeit* bezeichnet.

Systemänderungen dynamisch, online und automatisiert durchgeführt werden. Bezüglich des Aspekts, wie Adaptionen durchgeführt werden, unterscheidet [MSKC04b] zwischen *parametrischer* und *kompositionsbasierter* Adaption. Im Fall von parametrischer Adaption werden Variablen verändert, die Einfluss auf das Systemverhalten haben. Ein prominentes Beispiel für eine solche Adaption findet sich im *Transmission Control Protocol* (TCP) wieder [Pos81]. Hier wird das Verhalten in Bezug auf die aktuelle Netzwerkauslastung adaptiert, indem Parameter verändert werden, die das Empfangsfenster für Daten anpassen. Parametrische Adaptionen lassen sich in der Regel relativ einfach umsetzen. Dafür ist aber auch die Fähigkeit, auf a priori unbekannte Ereignisse reagieren zu können, beschränkt. Bei der kompositionsbasierten Adaptivität [MSKC04a] werden Algorithmen oder Komponenten des Systems ausgetauscht. Dadurch können Teile des Systems dynamisch zur Laufzeit rekonfiguriert werden, sodass sich das System an die gegebene Situation anpassen kann, wodurch sich die Flexibilität des Systems erhöht. Allerdings erfordert diese Art der Adaptivität einen Mehraufwand im Hinblick auf die Entwicklung des Systems. In Teilen der Literatur wird kompositionsbasierte Adaptivität auch als architekturbasierte Adaptivität bezeichnet (z.B. [GCH+04]).

Zusätzlich zu den bereits diskutierten Aspekten der Adaptivität von verteilten Systemen ist noch von Bedeutung, wie viel von der benötigten Adaptivität bereits zur Entwicklungszeit bekannt ist, sodass ein System entsprechend darauf vorbereitet werden kann. Hierbei lässt sich nach [Gei08] zwischen *antizipierter* und *nicht antizipierter* Adaptivität unterscheiden. Im Fall der ersten Form sind zur Entwicklungszeit alle Situationen bekannt, auf die das System in der Lage sein muss zur Laufzeit zu reagieren. Hingegen sind bei nicht antizipierter Adaptivität die möglichen Variationen zur Entwicklungszeit nicht bekannt. Daher müssen für diesen Fall Methoden implementiert werden, die zur Laufzeit in der Lage sind zu entscheiden, *welche* Adaptionen *wie* vorgenommen werden soll. Damit handelt es sich in diesem Fall um eine der komplexesten Formen der Adaptivität.

## 2.2. Selbst-Adaptive und Selbst-Verwaltende Systeme

Der dringende Bedarf zur dynamischen Adaption in Software-Systemen wird von einer breiten Forschungsgemeinschaft ausgedrückt. Diese Arbeiten bieten alternative Techniken zur Selbstorganisation, die es erlauben, adaptive, verteilte Softwaresysteme zu realisieren. Dabei werden eine Reihe von Terminologien verwendet, um die gemeinsame Vision von adaptiven Softwaresystemen zu beschreiben, die in der Lage sind, sich zur Laufzeit selbstständig anzupassen. Die prominentesten Bezeichnungen sind hierbei *self-adaptive* [ST09], *autonomic* [KC03] und *self-managing* [MWJ+07]. Im Folgenden werden wichtige Aspekte untersucht und vorgestellt, die konzeptionelle Alternativen zur Konstruktion von adaptiven, verteilen Systemen bieten. Eine detaillierte Untersuchung findet sich z.B. in [HM08, ST09] Die folgenden Abschnitte

befassen sich mit Klassifizierungsansätzen, um verschiedene Formen von verteilten Systemen zu unterscheiden, die adaptive Fähigkeiten aufweisen. Dabei wird deutlich werden, inwiefern sich selbstorganisierende Systeme mit anderen Ansätzen zur Adaption in verteilten Systemen überschneiden bzw. unterscheiden.

### 2.2.1. Dimension adaptiver Systeme

Adaptivität, wie sie in Abschnitt 2.1.3 beschrieben wurde, ist eine wünschenswerte Eigenschaften moderner verteilter Systeme. In [ALMW09] wird ein Klassifizierungsschema vorgeschlagen, das die Modellierung adaptiver Systeme in vier Dimensionen unterteilt. Somit ist es möglich, die Umsetzungen unterschiedlicher Konzepte für selbstadaptive Systeme besser vergleichen zu können. Im Folgenden werden die vier von [ALMW09] identifizierten Dimensionen vorgestellt:

**Ziele** sind globale Vorgaben, die ein System zu erreichen hat. Sie lassen sich wiederum untergliedern in die Anzahl der unterschiedlichen Ziele, die es gibt, ihren Zeithorizont, in welchem sie Abhängigkeiten zueinander aufweisen, und schließlich der Grad ihrer Flexibilität.

**Änderungen** sind die Ursachen, die eine Adaption auslösen. Dabei kann unterschieden werden, ob es sich bei ihnen um interne oder externe Ereignisse handelt. Außerdem ist die Frequenz der Änderungen von Bedeutung und ob es möglich ist, sie zu antizipieren oder nicht.

**Mechanismen** setzen die Reaktionen eines Systems auf Änderungen, die im Laufe des Adaptionsprozesses auftreten, um. Mechanismen lassen sich ebenfalls in verschiedene Unterkategorien einteilen. Dabei lässt sich unterscheiden, ob sie komplett autonom ausgeführt werden oder manuelle Eingriffe benötigt werden, ob sie einen lokalen oder globalen Geltungsbereich haben, ob die Adaptation auf Anpassungen von Systemparametern oder der Struktur abzielt und schließlich ob die Zeitdauer eine Rolle spielt.

**Effekte** beschreiben schließlich die Auswirkungen der Adaptation auf das System. Hierbei lässt sich weiter dahingehend charakterisieren, inwieweit eine gescheiterte Adaption für das System kritisch ist, ob die Auswirkungen der Adaption deterministisch vorhersehbar sind und inwiefern der Mehraufwand, der für eine Adaption nötig wird, die eigentliche Systemfunktionalität einschränkt.

Abschließend lässt sich feststellen, dass die Klassifizierung von [ALMW09], auf Basis der verschiedenen Dimensionen, eine feinere Unterscheidung und Charakterisierung selbstadaptiver Systeme im Hinblick auf ihre Adaptivität ermöglicht. Außerdem liefert sie eine einheitliche Terminologie und somit einen wichtigen Beitrag zum einheitlichen Verständnis von selbstadaptiven Systemen.

### 2.2.2. Hierachie adaptiver Systemeigenschaften

Die im vorangegangenen Abschnitt beschriebene Klassifizierung bzw. Dimensionierung liefert bereits erste Ansatzpunkte zur Unterscheidung von Konzepten, die verteilte Systeme mit adaptiven Eigenschaften ausstatten. Darauf aufbauend wird im Folgenden eine Hierarchie vorgestellt, welche weitere Unterscheidungsmerkmale im Hinblick auf die Adaptivität von verteilten Systeme liefert. Bei näherer Betrachtung im Bezug auf die Fähigkeit zur Adaptation wird schnell deutlich, dass diese nicht durch eine einzelne Komponente im System erbracht werden kann, sondern unterschiedliche Systemkomponenten auf unterschiedlichen Ebenen notwendig werden, um in ihrem Zusammenspiel Adaptivität zur Verfügung zu stellen. In diesem Kontext wurde in [ST09] eine Hierarchie von adaptiven Systemeigenschaften aufgestellt (siehe Abbildung 2.2).

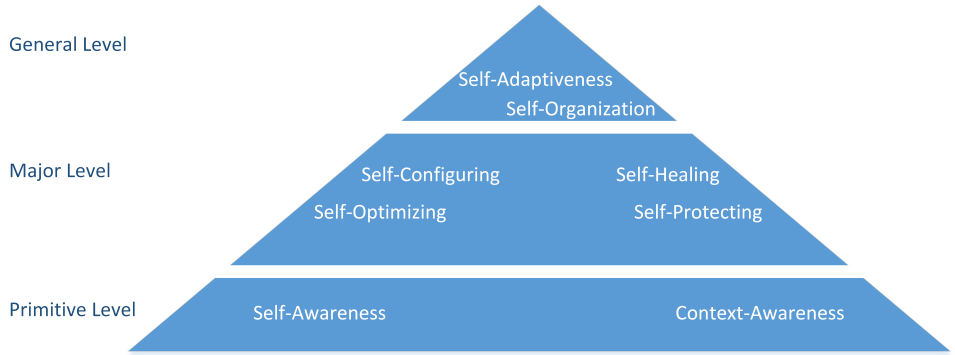



**Abbildung 2.2.:** Hierarchie adaptiver Systemeigenschaften - nach (ST09).

Die Hierarchie von adaptiven Eigenschaften nach [ST09] lässt sich in drei wesentliche Ebenen einteilen. Auf der untersten Ebene sind die grundlegenden Eigenschaften aufgeführt, die ein System besitzen muss, um adaptiv zu sein. Dazu zählen [ST09] *Self-Awareness* und *Context-Awareness*:

**Self-Awareness** bedeutet dabei, dass ein System in der Lage ist, sich selbst, insbesondere seinen Zustand und sein Verhalten, wahrzunehmen (siehe auch [HS06]). Eine grundsätzliche Voraussetzung hierfür sind Sensoren, die durch Selbstüberwachung das System in die Lage versetzen, sich selber zu beobachten.

**Context-Awareness** beschreibt in diesem Zusammenhang die Fähigkeit, den Kontext, in dem sich das System befindet, wahrzunehmen [PH05]. Der Kontext beschreibt dabei die Informationen, die dazu genutzt werden können, die Situation einer Entität zu charakterisieren [Dey01].

Wenn ein verteiltes System die adaptiven Eigenschaften der untersten Ebene besitzt, dann ist es fähig, Eigenschaften zur Verfügung zu stellen, die sich in der darüber liegenden Ebene befinden:

**Self-Configuring** bezeichnet die Fähigkeit eines Systems, sich automatisch umzukonfigurieren, wenn Änderungen im System oder der Umgebung

dieses erfordern. Die Rekonfiguration beinhaltet dabei z.B. die Installation, das Update, das Zusammensetzen oder das Abtrennen von Softwarekomponenten.

**Self-Healing** ist eine adaptive Eigenschaft, die die Fähigkeit bezeichnet, Störungen im System zu entdecken, zu diagnostizieren und angemessen darauf zu reagieren. Dazu gehört es auch, potenzielle Probleme zu antizipieren und durch die Einleitung vorbeugender Maßnahmen zu vermeiden. In der Literatur wird diese Eigenschaft auch als *Self-Diagnosing* [RL05] oder *Self-Repairing* [dLF02] bezeichnet.

**Self-Optimising** bezeichnet die Fähigkeit eines Systems, die Performanz und Allokation von Ressourcen so bereitzustellen, dass unterschiedliche Nutzeranforderungen erfüllt werden können. Beispiele hierfür sind u.a. der Durchsatz oder die Reaktionszeit. Self-Optimisation wird in der Literatur auch im Zusammenhang mit *Self-Tuning* bzw. *Self-Adjusting* genannt [SPTU05].

**Self-Protecting** schließlich bezeichnet die adaptive Eigenschaft, dass ein System in der Lage ist, Bedrohungen bezüglich seiner Sicherheit zu erkennen und angemessen zu reagieren. Hierbei geht es also darum, ein System proaktiv gegen Angriffe zu schützen. Dabei gilt es, mögliche Gefahren zu antizipieren und im Fall eines erfolgten Angriffs diesen einzudämmen bzw. zu beseitigen.

Diese und ähnliche Eigenschaften werden in der Literatur als *Self-** Eigenschaften von adaptiven Systemen zusammengefasst [KC03, BJM+05]. Sie versetzen ein System in die Lage, sich selbst zu verwalten. Daher bilden diese Eigenschaften die Grundlage für die oberste, generelle Ebene (siehe Abbildung 2.2). Auf dieser Ebene werden adaptive Systeme anhand ihrer globalen Eigenschaften unterschieden. Das wesentliche Unterscheidungsmerkmal hierbei ist die Architektur des Systems:

**Self-Adaptiveness:** Basiert das System auf einer hierarchischen Architektur mit einer zentralen Kontrollinstanz, so gehört es nach [ST09] zur Klasse der *selbstadaptiven* Systeme. Die Vorgehensweise zum Erreichen des adaptiven Verhaltens wird bei dieser Art von Systemen global vorgegeben und durch die unteren Systemkomponenten umgesetzt. Selbstadaptive Systeme beruhen somit auf einer *Top-down* Modellierung.

**Self-Organization:** Wenn die Architektur hingegen auf einem dezentralen Ansatz mit gleichberechtigen Komponenten beruht und keine zentrale Kontrollinstanz aufweist, so gehört das System nach [ST09] zur Klasse der *selbstorganisierenden* Systeme. Für diese Klasse ist es charakteristisch, dass die Dynamik, die zu dem adaptiven Verhalten führt, von den unteren Systemkomponenten erzeugt wird und sich dann auf der obersten, globalen Ebene äußert. Selbstorganisierende Systeme basieren daher nach [ST09] auf einer *Bottom-up* Herangehensweise.

Beide Ansätze zielen dabei auf die Bereitstellung automatisierter Laufzeitadaptivität ab, wählen aber grundsätzlich unterschiedliche Ansätze. Im Fokus dieser Arbeit stehen selbstorganisierende Systeme. Da jedoch auch selbstadaptive Systeme ein großes Interesse in der Wissenschaft und Wirtschaft geweckt haben, werden im Folgenden zwei prominente Ansätze exemplarisch vorgestellt werden. *Autonomic Computing* ist dabei ein international weit verbreiteter Ansatz, während es sich bei *Organic Computing* um ein eher national verbreitetes Konzept handelt.

### 2.2.3. Autonomic Computing

Autonomic Computing basiert auf einem Manifest, das IBM im Oktober 2001 veröffentlichte [Res01]. In diesem Manifest proklamierte IBM die Vision für zukünftige, moderne Computersysteme. Im Kern wurde dabei die These vertreten, dass nicht mehr, wie es bis dahin größtenteils der Fall war, mangelnde Performanz der Hard- und Software-Komponenten die Weiterentwicklung von IT-Systemen verhindern, sondern vielmehr die wachsende Komplexität dieser Systeme zur Herausforderung wird. Dieser Trend geht einher mit dem rasanten Wachstum des Internets und der damit wachsenden Bedeutung von *E-Business*, also Wertschöpfungsprozessen in der Wirtschaft, die in großem Maße auf Informationstechnologie beruhen. IBM hat dies am Beispiel von Geldautomaten verdeutlicht [GC03]. Gab es früher einige zehntausende Automaten mit relativ fixer Funktionalität, die über nichtöffentliche Netzwerke verbunden waren, so können heute sehr vielfältige Finanztransaktionen durch verschiedene Endgeräte (Personal Computer, Tablets, Smartphones etc.) von mehreren Millionen Menschen weltweit über das Internet getätigt werden. Diese Entwicklung verdeutlicht, wie das Management von Computersystemen vor neue Herausforderungen gestellt wird. Vor diesem Hintergrund präsentierte IBM das Paradigma des *Autonomic Computing*. Es sollte die Entwicklung von Computersystemen forcieren, die in der Lage sind, sich selbstständig zu verwalten, um damit die Abhängigkeiten von Systemadministratoren zur Laufzeit zu minimieren. Als Inspirationsquelle diente daher u.a. das menschliche Nervensystem, welches grundlegende Funktionalitäten des menschlichen Körpers automatisch bewältigt, ohne dass eine explizite kognitive Steuerung durch den Menschen notwendig wird. Dabei entlastet das Nervensystem die kognitiven Fähigkeiten des Menschen und erlaubt es diesem, sich auf globale Ziele zu fokussieren, während Teile der tatsächlich damit verbundenen Umsetzung unterbewusst geschehen. Als Beispiel sei hierbei eine Person genannt, die zur U-Bahn läuft, um nicht zu spät zu einem wichtigen Termin zu kommen. Dabei werden automatisch vom Nervensystem die Atmung und die Herzfrequenz angepasst, um auf die neue Situation, den Wechsel vom Geh- zum Laufschritt, zu reagieren. Analoges Verhalten dazu wird durch das *Autonomic Computing* Paradigma für moderne, verteilte Systeme anvisiert.

Neben der zunehmenden Komplexität von IT-Systemen, spielen auch Kosteneinsparungen eine entscheidende Rolle dafür, dass wichtige IT-

Unternehmen an der Umsetzung von *Autonomic Computing* interessiert sind [MH04]. Dabei geht es den Unternehmen darum, Kosten für den Betrieb und die Wartung ihrer IT-Infrastrukturen einzusparen, indem Administratoren nur noch Ziele und Richtlinien auf höherer Abstraktionsebene vorgeben müssen und die Systeme die Umsetzung automatisch übernehmen. Dabei verdeutlichen folgende Fakten aus [GC03] die Gründe für die Durchsetzung von *Autonomic Computing*:

- Um IT-Systeme vor deren Absturz zu schützen oder sie gegebenenfalls wiederherzustellen, setzen Firmen schätzungsweise zwischen 33%-50% ihres gesamten IT-Budgets ein.
- Das Verhältnis von Ausgaben für Computerspeicher zu dessen Administration liegt bei 1 zu 9.
- 60%-70% der gesamten Kosten für Datenbanksysteme entfallen auf deren Administration, d.h. die Installation, Wartung und Gehälter für Administratoren.

Um nun Computersysteme zu realisieren, die die gewünschten Eigenschaften zur Selbstverwaltung erfüllen, werden im ursprünglichen *Autonomic Computing* Manifest von IBM [Res01] acht Eigenschaften genannt, die diese Systeme auszeichnen:

1. Damit ein Computersystem autonom handeln kann, muss es eine explizite abstrakte Repräsentation seiner selbst besitzen. Dieses gilt ebenfalls für die Komponenten des Systems.
2. Ein *Autonomic Computing* System muss in der Lage sein, sich selbst zu konfigurieren. Zudem muss es sich unter veränderlichen und nicht vorhersehbaren Bedingungen rekonfigurieren können.
3. Ein *Autonomic Computing* System gibt sich nie mit dem Status quo zufrieden. Es sucht immer nach weiteren Optimierungsmöglichkeiten.
4. Ein *Autonomic Computing* System muss in der Lage sein, sich selber zu heilen. Das bedeutet, dass es auf Ereignisse reagieren muss, die im System Funktionsstörungen verursachen.
5. Da es in der Computerwelt, ebenso wie in der realen Welt, Gefahren gibt, muss ein *Autonomic Computing* System gute Fähigkeiten besitzen, sich selbst zu verteidigen.
6. Ein *Autonomic Computing* System besitzt eine Repräsentation und Kenntnis über seine Umwelt. Diese Fähigkeiten nutzt es zusammen mit der Wahrnehmung von Aktivitäten, um sich entsprechend zu verhalten.
7. Ein *Autonomic Computing* System kann nicht in einer hermetisch abgeschlossenen Umwelt existieren.

8. Ein *Autonomic Computing* System versucht, die optimale Bereitstellung von Ressourcen zu antizipieren, während es gleichzeitig die damit verbundene Komplexität für den Benutzer transparent[2] hält.

Computersysteme, die diese Eigenschaften besitzen werden auch als selbstverwaltende Systeme beschrieben [MWJ+07]. Außerdem zeigt es sich, dass diese Eigenschaften auch als Self-* Eigenschaften verteilter Systeme verstanden werden können (vgl. Abschnitt 2.2.2). Um verteilte Systeme zu realisieren, die die erwähnten Eigenschaften zur Selbstverwaltung bereitstellen, wird in [KC03] ein Architekturreferenzmodell vorgeschlagen. Im Kern sieht das Modell vor, dass die Ressourcen eines Computersystems durch sogenannte *Autonomic Manager* gesteuert werden. Ressourcen bezeichnen in diesem Kontext beliebig komplexe Entitäten eines Computersystems, z.B. eine einzelne Festplatte, eine Datenbank oder auch ein komplettes Alt-System. Ressourcen, die von einem *Autonomic Manager* verwaltet werden, werden dabei als *managed elements* (verwaltete Elemente) bezeichnet. Der *Autonomic Manager* durchläuft dabei zur Steuerung der ihm zugeordneten Ressourcen die sogenannte *MAPE-K* Schleife (siehe Abbildung 2.3). Die Schleife beginnt mit dem *Monitor*ing der verwalteten Ressource; wenn dort ein Ereignis oder ein Zustand detektiert wird, der möglicherweise eine Änderung erfordert, so wird dieses im folgenden *Analyse*-Schritt näher betrachtet. Sollte das Ereignis eine Änderung an der Ressource erfordern, so wird mit Hilfe der *Plan*ungskomponente zunächst auf abstrakter Ebene ein Ausführungsplan entworfen. Dieser kann dann in dem folgenden Schritt von einer weiteren Komponente ausgeführt (engl. „to *execute*“) werden, was eine Umsetzung in eine tatsächliche Aktion zur Folge hat. Ergänzt wird die gesamte Schleife durch eine Wissenskomponente (engl. *Knowledge*), auf die alle Teilkomponenten Zugriff haben, um dort abgespeicherte Regeln, Strategien usw. abzurufen und in ihre eigene Verarbeitung einzubinden. Durch die Interaktion von einzelnen, durch einen *Autonomic Manager* verwalteten Ressourcen mit anderen Ressourcen und die Bildung von Hierarchien können so komplexe Kontrollstrukturen realisiert werden, die es verteilten Systemen erlauben, sich selbstständig zu verwalten. Die Bildung von Hierarchien ist somit ein wesentliches Merkmal dieses Ansatzes, da erst dadurch komplexe Computersysteme modelliert werden können, wie sie auch in heutigen Anwendungslandschaften vorzufinden sind. Konkrete Implementierungen der MAPE-K Referenzarchitektur werden z.B. mit der *Agent Building and Learning Enviroment (ABLE)* [BSP+02] und *Kinesthetics eXtreme* [KPGV03] beschrieben.

### 2.2.4. Organic Computing

*Organic Computing* entstand aus dem Schwerpunktprogramm 1183 der Deutschen Forschungsgemeinschaft (DFG). Die Grundidee ist, ähnlich wie bei *Autonomic Computing* flexible und autonome IT-Systeme zu bauen, die

[2] Transparenz bedeutet im Sinne der Informatik, dass Vorgänge für Außenstehende nicht sichtbar sind bzw. die mit den Vorgängen verbundenen Handlungen verborgen bleiben.

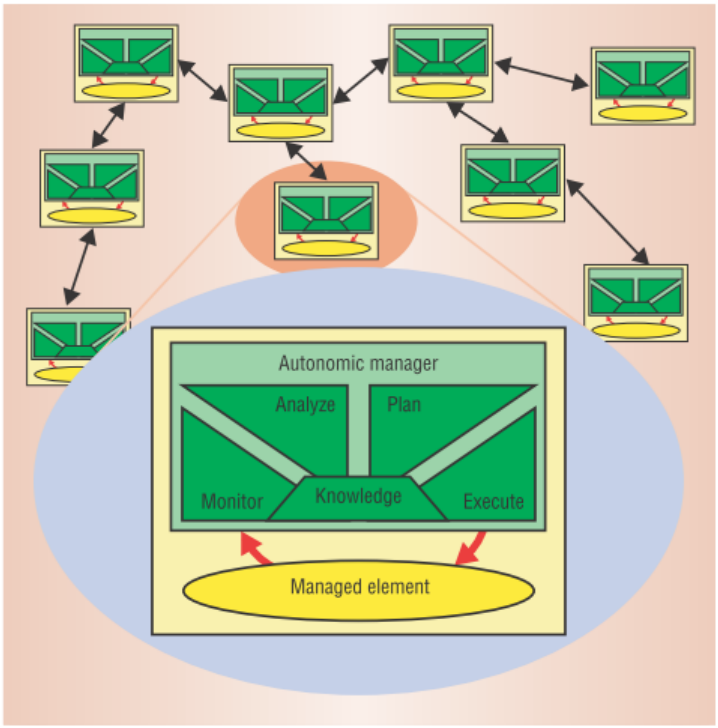



**Abbildung 2.3.:** Architektur eines verteilten Systems mit Ressourcen, die durch einen Autonomic-Manager auf Basis der MAPE-K Schleife gesteuert werden - aus (KC03, S. 44).

sich an etablierten Konzepten und Strukturen komplexer Organismen orientieren [MS04, BMMSP06]. Im Gegensatz zum *Autonomic Computing*, wo es mit dem zentralen Nervensystem des Menschen eine relativ eindeutige Inspirationsquelle gibt, orientiert sich *Organic Computing* allgemeiner an Organismen, die ein selbstorganisierendes und/oder emergentes Verhalten besitzen. Ein weiterer Unterschied liegt in den Anwendungsdomänen der Forschungsinitiativen. Während *Autonomic Computing* darauf abzielt, bestehende IT-Systeme mit minimalen Anpassungen in die Lage zu versetzen, sich selbst zu verwalten (Evolution), legt *Organic Computing* den Fokus auf die Entwicklung neuartiger Systeme [VDE03]. Im Fokus stehen dabei sowohl die Hardware als auch die Software, beide sollen im Zuge des *Organic Computing* Designs auf Basis identifizierter Konzepte in organischen Lebewesen zu einem neuartigen Computer werden.

Im Zentrum des *Organic Computing* Ansatzes steht die *Observer/Controller Architektur* [BMMSP06], die das wesentliche Aufbaumuster organischer Computerkomponenten beschreibt (siehe Abbildung 2.4). Das Architekturmuster basiert auf dem Paar *Observer* (Beobachter) und *Controller* (Kontrolleur), dass das System beobachtet und Situationen im Hinblick auf übergeordnete Ziele überprüft, um ggf. aktiv in das System einzugreifen. Ein Eingriff findet immer dann stand, wenn das System einen Zustand erreicht, der aus globaler Sichtweise nicht wünschenswert ist bzw. die Erfüllung der Systemziele gefährdet. Abgesehen davon hat das System die Freiheit, sich selber zu orga-

nisieren. Diese Architektur zielt darauf ab, Systeme mit selbstorganisierenden und/oder emergenten Verhalten zu unterstützen. Weil die Entwicklung solcher Systeme aber auf einem Bottom-up Ansatz basiert, soll gleichzeitig durch das Observer/Controller Paar eine Top-down Entwicklung unterstützt werden, die auf oberster Ebene sicherstellt, dass die Systemziele erreicht werden [MS04]. Dadurch soll es zu einer Kontrolle des selbstorganisierenden Verhaltens kommen [RMB+06].

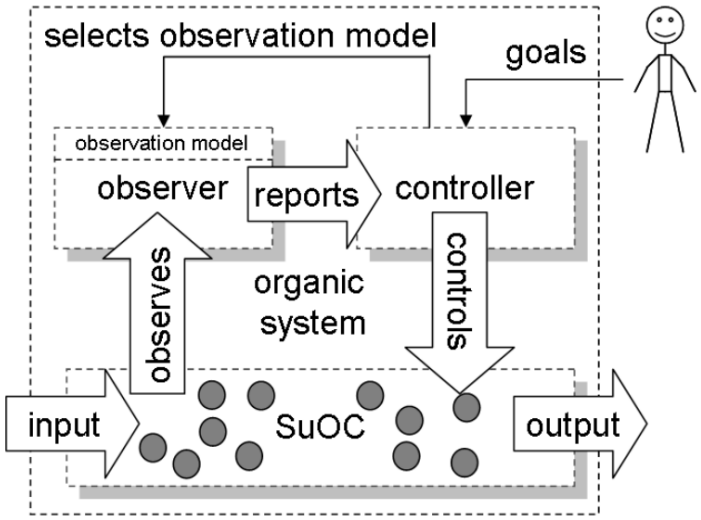



**Abbildung 2.4.:** Observer/Controller Architektur des *Organic Computing* - aus (BMMSP06, S. 186).

Abschließend lässt sich feststellen, dass die Vision des *Organic Computing* ähnlich dem *Autonomic Computing* im Kern einen Ansatz verfolgt, der auf einer zentralen Steuerung des Systems beruht. Dieses basiert auf einem Top-down Entwicklungsansatz, der globale Ziele und Regeln für das System vorgibt und diese durchsetzt.

### 2.2.5. Unterscheidungsmerkmale adaptiver Systeme

In den vorherigen Abschnitten wurde diskutiert, welche unterschiedlichen Facetten es im Kontext der Bereitstellung und Befähigung von Adaptivität in verteilten Systemen gibt. Im Rahmen der Dissertation von Vilenica [Vil13] wurden wichtige Unterscheidungsmerkmale für verteilte Systeme mit der Fähigkeit zur Laufzeitanpassung ausgearbeitet. Diese sind in Abbildung 2.5 dargestellt und unterscheiden zwischen folgenden drei charakteristischen Merkmalen:

**x-Achse:** Beschreibt den Aufbau der internen Systemarchitektur. Ist diese hierarchisch (zentral) oder dezentral aufgebaut?

**y-Achse:** Beschreibt, wie die Adaptivität zur Laufzeit umgesetzt wird. Dieses kann manuell, d.h. von einem Systemadministrator umgesetzt oder automatisiert vom System selber umgesetzt werden.

**z-Achse:** Beschreibt, welche Änderungen am System vorgenommen werden, um Adaptivität zu erreichen. Dabei wird zwischen einzelnen Systemparametern und Änderungen an den Systemkomponenten bzw. dessen Struktur unterschieden.

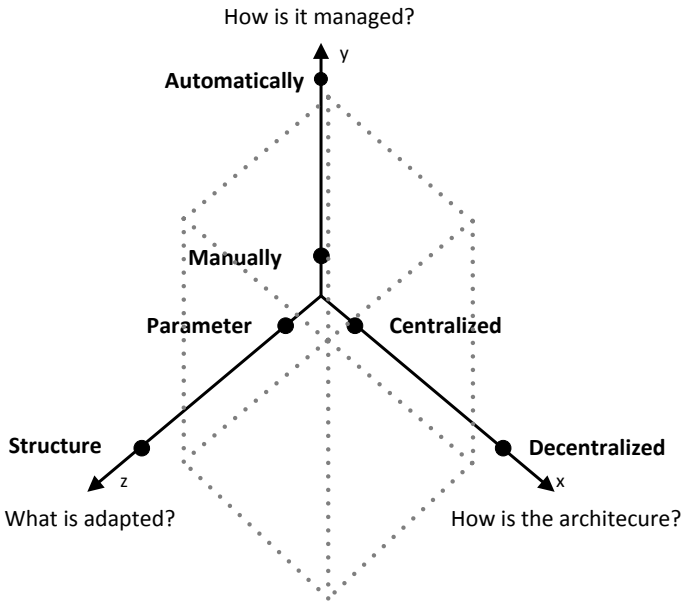



**Abbildung 2.5.:** Charakteristische Unterscheidungsmerkmale für verteilte Systeme mit der Fähigkeit zur Laufzeitanpassung - aus (Vil13, S. 26).

In den vorangestellten Abschnitten wurde innerhalb dieser Unterscheidungsmerkmale ein möglichst hoher Grad an Automatisierung als erstrebenswert anerkannt. Mit den selbstadaptiven Systemen wurde dabei eine Klasse identifiziert, die diese Anforderung erfüllt, wobei die Lösungskonzepte hauptsächlich auf zentralen Architekturformen basieren. Exemplarisch wurden mit den Konzepten des *Autonomic Computing* und *Organic Computing* zwei Ansätze aus dieser Klasse von Systemen vorgestellt. Im Folgenden wird eine Klasse von Systemen vorgestellt werden, die automatisierte Laufzeitanpassungen auf Basis dezentraler Architekturformen unterstützt.

## 2.3. Selbstorganisation auf Basis dezentraler Koordination

Während die im Abschnitt 2.2 betrachteten selbstadaptiven Systeme die gewünschte Adaptivität auf Basis zentraler Komponenten und hierarchischer Strukturen umsetzen, ist Dezentralität und die damit einhergehende Abwesenheit zentraler Kontrollstrukturen charakteristisch für selbstorganisierende Systeme. Daraus resultiert die Herausforderung, wie in einem solchen selbstorganisierenden System ein kohärentes adaptives Verhalten erzielt werden kann. Im Rahmen dieses Abschnitts wird herausgearbeitet werden, wie Koordination verwendet werden kann um das gewünschte Verhalten zu erreichen.

### 2.3.1. Konzeption

Die Bedeutung von Koordination für verteilte selbstorganisierende Systeme kann mit Hilfe des in Abbildung 2.6 dargestellten Modells beschrieben werden. In diesem konzeptionellem Modell [SR10b, Sud10] werden verteilte Systeme, Adaptionsdynamik und Koordination in Beziehung gesetzt.

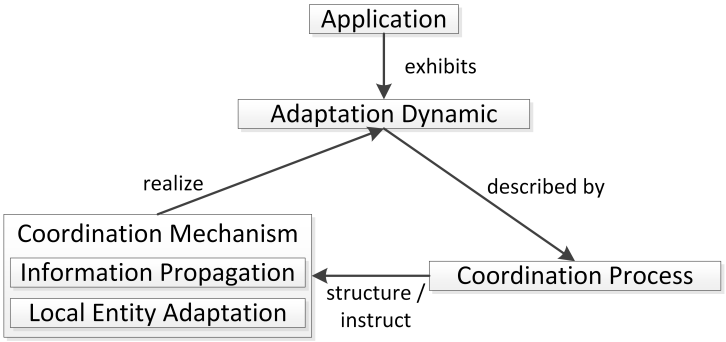



**Abbildung 2.6.:** Konzept von Adaptionsdynamik, Koordinationsprozess und Koordinationsmechanismus in einer verteilten Anwendung - nach (Sud10).

Gemäß dem Modell lässt sich in einem verteilten System ein Verhalten beobachten, das auf Systemebene für eine adaptive Dynamik sorgt. Diese Adaptationsdynamik lässt sich wiederum durch einen Koordinationsprozess beschreiben. Dieser Prozess ist dafür verantwortlich, dass sich das verteilte System kontinuierlich selbst organisiert, indem es sich neu strukturiert bzw. sein Verhalten adaptiert. Ein Koordinationsprozess dient dabei als Strukturmuster, das die Abhängigkeiten zwischen den einzelnen Komponenten im System definiert und überwacht. Dabei beeinflusst bzw. strukturiert der Koordinationsprozess das Verhalten der einzelnen Systemkomponenten auf dezentrale Weise. Diese Fähigkeit beruht maßgeblich auf der Verwendung von dezentralen Koordinationsmechanismen. Diese Mechanismen erfüllen wiederum zwei Aufgaben. Sie stellen sowohl Techniken zum Informationsaustausch (*Information Propagation*) zwischen den Systemelementen zur Verfügung als auch Techniken, welche die Anpassung der lokalen Systemelemente ermöglichen (*Local Entity Adaptation*). Durch die Verwendung und Kombination der Koordinationsmechanismen wird schließlich eine Dynamik erzeugt, die sich auf oberer Systemebene als adaptives Verhalten beobachten lässt. Die Adaptivität in diesem Modell beruht dabei auf dem kohärenten Verhalten der lokalen Systemelemente, welches durch dezentrale Koordination erzielt wird. Damit unterscheidet sich der Ansatz von selbstadaptiven Modellen, bei denen das adaptive Verhalten von zentralen Kontrollinstanzen durchgesetzt wird. Dieses in [SR10b, Sud10] vorgestellte Modell beschreibt auf einer hohen Abstraktionsebene ein Verständnis von selbstorganisierenden Systemen, welches auf dezentraler Koordination beruht. Zudem wird an dieser Stelle erneut die Abgrenzung zwischen selbstorganisierenden und selbstadaptiven Systemen deutlich.

### 2.3.2. Koordination, Kommunikation und Interaktion

Zum besseren Verständnis der Funktionsweise von selbstorganisierenden Systemen, die auf Koordination beruhen, soll an dieser Stelle eine genauere Betrachtung der Begriffe Koordination, Kommunikation und Interaktion erfolgen. Die Begriffe Koordination und Kommunikation werden in der Literatur unterschiedlich, abhängig vom jeweiligen Kontext, definiert. Eine Definition des Begriff der Koordination aus der Soziologie von *Aschke* definiert Koordination als Einheit der externen Selektion:

> *Unter dem Begriff der Koordination werden Handlungskomplexe unterschiedlichster Art (z.B. Interaktionen, Märkte, Organisation, Institutionen) verstanden bei denen die individuellen Akteure ihre Handlungsbeiträge dadurch aufeinander abstimmten, daß sie sich bei der Wahl ihrer Handlungen maßgeblich an Information orientieren, die in Kommunikationssystemen erzeugt und weitergegeben werden.*
>
> [Asc02, Seite 338]

Bei der Betrachtung der Definition von Koordination ist auffällig, dass das Abstimmen von Handlungen bzw. Aktivitäten zwischen mehreren Partnern oder Akteuren im Vordergrund steht. *Aschke* geht in seiner Definition dabei darauf ein, dass bei der Wahl der Handlungen Informationen, die in Kommunikationssystemen erzeugt oder weitergeben werden, eine maßgebliche Rolle spielen. Koordination setzt also Kommunikation voraus. Kommunikation wird dementsprechend auch im Rahmen der Informationstheorie von *Shannon* als Übertragung von Informationen zwischen Partnern definiert [Sha48]. Eine weitere Abgrenzung der Begriffe Koordination und Kommunikation aus dem Umfeld der *language/action perspective* (LAP) wird in [WvdPdM03] gegeben. Diese Arbeit greift auf vorangegangene Arbeiten von *Thomas W. Malone* und *Kevin Ghen Crowston* [MC94] zurück, die 1994 die interdisziplinäre Wissenschaft der Koordination vorgeschlagen haben. *Malone* und *Crowston* definieren Koordination als:

> *Coordination is the act of managing interdependencies between activities performed to achieve a goal.*
>
> [MC94]

Der Bedarf für Koordination einsteht also aus der Existenz von Abhängigkeiten. Mit Hilfe von Koordination können diese Abhängigkeiten zwischen Aktivitäten bzw. Systemkomponenten verwaltet werden. Koordination ist somit das Bindemittel, das separate und eigenständige Komponenten zu einer Gesamtheit formt [GC92]. Diese Auffassung wird von *Guinea und Saeedi* [GS12] ergänzt um das Erreichen eines gemeinsamen Ziels, wobei Koordination eine Organisationsfunktion übernimmt. In diesem Kontext ordnet [Bra07] Koordination zwei primäre Funktionen zu: In Bezug auf das Gesamtsystem erhöht sie die Effektivität und regelt dabei mögliche Konflikte. Dies bedeutet aber nicht zwangsläufig, das alle beteiligten Akteure kooperativ sein müssen; [HS99] betont, dass Koordination nicht nur im kooperativen Sinn, sondern

auch im Wettbewerb eingesetzt werden kann. Dieses Verständnis von Koordination bedeutet auch, dass, wenn keine Abhängigkeiten zwischen Systemkomponenten bestehen, es auch keinen Bedarf an Koordination gibt [MC94]. Für die Betrachtung von selbstorganisierenden Systemen kann i.d.R. von einem hohen Bedarf an Koordination ausgegangen werden, da diese Systeme meistens aus vielen einfachen Systemkomponenten bestehen, die einen hohen Grad an Interdependenz aufweisen.

Die in der Koordinations-Definition von *Aschke* [Asc02] erwähnte Verwendung von Kommunikation zur Koordination lässt sich noch weiter in unterschiedliche Interaktionsarten unterscheiden. Allgemein lässt sich feststellen, dass Koordination auf unterschiedlichen Interaktionsmustern beruhen kann, um Abhängigkeiten und das Verhalten von Systemkomponente zu beeinflussen. In der Dissertation von *Pokahr* [Pok07] werden unterschiedliche Arten der Interaktion und ihr Einfluss auf die Koordination eines Multi-Agenten Systems beschrieben. Abbildung 2.7 leitet basierend auf der Beschreibung in [Pok07] eine Taxonomie möglicher Interaktionsmuster ab. Dabei wird zunächst unterschieden, ob die Interaktion auf direkter oder indirekter Form von Kommunikation basiert oder sogar komplett ohne Kommunikation stattfindet. Interaktion ohne jegliche Kommunikation kann z.B. durch Interaktion der Entitäten mit ihrer Umwelt erfolgen; dies wird auch *Stigmergie* genannt.

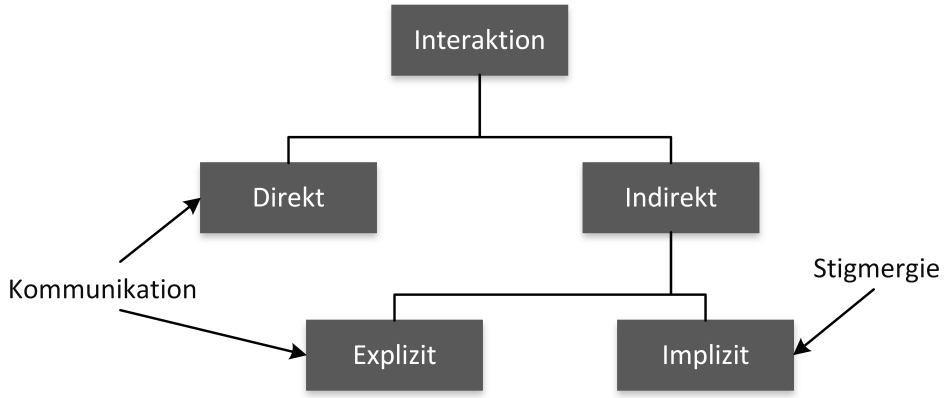



**Abbildung 2.7.:** Taxonomie unterschiedlicher Arten von Interaktion - nach (Pok07).

### Stigmergie

Stigmergie ist ein Kunstwort, welches sich aus dem griechischen Begriffen *„stigmata“* (Zeichen) und *„ergon“* (Arbeit bzw. Aufgabe) zusammensetzt [Brü00]. Das Prinzip der Stigmergie wurde bereits in den 50er Jahren des letzten Jahrhunderts vom französischen Zoologen Grassè [Gra59] beobachtet, als er das Nestbauverhalten von Termiten studierte. Stigmergie beschreibt dabei die Koordination einzelner Arbeiter durch externe Zeichen. Grundsätzlich können zwei Formen von Stigmergie unterschieden werden [HW94, Whi97]: Bei sematektonischer Stigmergie sind die Zeichen selbst ein Teil der Aufgabe (z.B. die Form eines Bogens in einem Termitenbau). In der zeichenbasierten

Stigmergie dienen zusätzliche Marken der Koordination zwischen den Individuen (z.B. Pheromone bei Ameisen). Bei der kommunikationslosen Interaktion durch Stigmergie benutzen die Entitäten also gezielt Gegenstände der Umgebung (zeichenbasiert) oder als Nebenprodukt ihr Tätigkeit (sematektinisch) zur gegenseitigen Beeinflussung. Ein Beispiel für sematektonische Stigmergie sind Interface-Agenten, die als persönliche Assistenten den Benutzer durch seine Interaktion mit der Anwendung wahrnehmen. Da dabei unerheblich ist, ob dem Benutzer bewusst ist, wie seine Aktionen durch den Agenten wahrgenommen werden, handelt es sich hierbei um sematektonische Stigmergie (vgl. [Pok07]). Mobile Agenten, die im Bereich des automatisierten Netzwerk-Routings Pheromone ablegen, um nach und nach optimale Routen durch das Netzwerk zu finden [SHB97] interagieren dabei auf Basis von zeichenbasierter Stigmergie. Bei dieser Form der kommunikationslosen Interaktion, gibt es keine dediziert nur für die Kommunikation zuständigen Elemente im System, vielmehr werden vorhandene Objekte im System genutzt, um sich darüber zu koordinieren. Daher wird diese Form der Koordination auch häufig in selbstorganisierenden Systemen beobachtet.

#### Direkte Kommunikation

Die direkte Kommunikation ist die gängigste Kommunikation in verteilten Systemen. Sie hat ihre Wurzeln einerseits in der Interprozesskommunikation und andererseits werden viele Anleihen bei der Sprachphilosophie aus dem Bereich der Linguistik gemacht. In verteilten Systemen wird direkte Kommunikation häufig basierend auf dem Austausch von Nachrichten realisiert. Dabei unterscheidet man zwischen synchroner und asynchroner sowie verbindungsorientierter und verbindungsloser Kommunikation. Bei der synchronen Kommunikation findet ein zeitlicher Abgleich zwischen dem Sender und dem Empfänger statt, d.h. der empfangende Prozess blockiert bis zum Eingang der Nachricht und der sendende Prozess blockiert, solange die Nachricht nicht entgegengenommen wurde. Bei der asynchronen Kommunikation sind Sender und Empfänger entkoppelt, in diesem Fall verfügt der empfangende Prozess über ein Postfach, in das eingehende Nachrichten abgelegt und bei Bedarf abgerufen werden. Der sendende Prozess kann eine Nachricht im Postfach des Empfängers ablegen und muss nicht warten, bis diese bearbeitet wurde. Bei der verbindungsorientierten Kommunikation werden zwischen den Kommunikationspartnern Kanäle aufgebaut, in denen die Daten als kontinuierliche, geordnete Ströme hin und her gesendet werden. In der verbindungslosen Kommunikation werden Datenpakete einzeln und unabhängig voneinander verschickt. Hierbei müssen die Sende- bzw. Empfangsereignisse nicht notwendigerweise in der Reihenfolge übereinstimmen.

#### Indirekte Kommunikation

Während bei der Stigmergie eine implizite Form der Interaktion stattfindet, bei der sich die beteiligten Komponenten nicht über die Interaktion be-

wusst sein müssen, findet bei der indirekten Kommunikation eine explizite Interaktion statt, bei der die Komponenten bewusst mit einem Kommunikationsmedium interagieren. Dabei beseht das Übertragungsmedium im Gegensatz zur Stigmergie nicht aus passiven Objekten, sondern ist zumeist selbst ein aktiver Prozess. Das bekannteste Beispiel für indirekte Interaktion sind *Blackboard-Systeme* [Cor91]. In solchen Systemen ist das *Blackboard* (engl. für Wandtafel) eine vermittelnde Komponente zwischen den verteilten Prozessen. Jeder Prozess hat dabei Zugriff auf das *Blackboard*, nicht aber auf die anderen Prozesse. Die Prozesse kommunizieren so indirekt, indem sie die relevante Information auf dem *Blackboard* publizieren. Das *Blackboard* vermittelt aktiv zwischen den Prozessen, indem es Änderungen, die für bestimmte Prozesse relevant sind, an diese weiterleitet. Zudem regelt es den Zugriff. Ein Beispiel für eine konkrete Umsetzung eines *Blackboard*-Systems bilden die *Coordination Artifacts* von *Ominici et al.* [ORV+04].

### 2.3.3. Koordinationsmechanismen

In diesem Abschnitt wird untersucht, welche Formen von Koordinationsmechanismen existieren, um Koordination in verteilten und selbstorganisierenden Systemen umzusetzen. Dabei wird zunächst eine allgemeine Klassifikation von Koordinationsmechanismen präsentiert, bevor Koordinationsmechanismen betrachtet werden, die speziell für selbstorganisierende Systeme von Bedeutung sind.

#### Klassifikation allgemeiner Koordinationsmechanismen

In [Bra07] wurde eine allgemeine Klassifikation von Koordinationsmechanismen präsentiert. Diese unterscheidet zunächst zwischen Szenarien, in denen es um die Koordination von kooperativen oder konkurrierenden Verhalten geht. Als weiteres Unterscheidungsmerkmal wird nach der Ebene der Anwendung der Koordinationsmechanismen differenziert. Dabei wird zwischen der Ebene des Individuums, der Gruppe und der (sozialen) Umgebung unterschieden. In Tabelle 2.1 werden einige Beispiele für Ausprägungen von Koordinationsmechanismen gemäß der Klassifizierung aus [Bra07] aufgeführt, die im Folgenden beschrieben werden.

**Tabelle 2.1.:** Klassifikation von Koordinationsmechanismen nach (Bra07).

| | **Individuum** | **Gruppe** | **Gesellschaft** |
|---|---|---|---|
| **Kooperation** | Antizipation<br>Argumentation | Teamarbeit | Soziale Strukturen<br>Soziale Normen |
| **Konkurrenz** | Antizipation | Marktmechanismen | Soziale Gesetze<br>Soziale Normen |

Auf der Ebene des Individuums gibt es grundsätzlich zwei Arten von Koordinationsmechanismen. Bei der Antizipation versucht eine Komponente

vorauszuahnen, welche Handlungen andere Komponenten in der Zukunft ausführen könnten. Das Antizipieren von Handlungen anderer Komponenten kann dazu verwendet werden, sich mit diesen abzustimmen. Hierbei ist es notwendig, dass eine Komponente in der Lage ist, sich in andere hinein zu versetzen, d.h. eine Vorstellung von anderen Komponenten hat, um deren Verhalten antizipieren zu können. In diesem Zusammenhang spricht man auch von *coordination by mutual modelling* [Bra07]. Koordinationsmechanismen auf Basis von Argumentation haben das Ziel, die Handlungen von anderen Komponenten dadurch zu beeinflussen, dass Teile der eigenen Sichtweise offengelegt werden. Dadurch gibt eine Komponente Informationen an die zu beeinflussende Komponente weiter, die diese zuvor nicht hatte. Somit wird klar, dass Argumentation im Gegensatz zur Antizipation nur für kooperative Szenarien geeignet ist.

Auf Ebene der Gruppe sind Koordinationsmechanismen dadurch charakterisiert, dass sie das Ziel verfolgen, Aktivitäten mit anderen Gruppenmitgliedern abzustimmen. In kooperativen Szenarien wird dabei die gemeinsame Arbeit an den für die Gruppe relevanten Aufgaben in den Vordergrund gestellt. In diesem Fall spricht man von Teamarbeit. Komponenten können dabei auf zwei unterschiedlichen Ebenen miteinander interagieren. Einerseits können sie sich dazu entschließen, an einer gemeinsamen Zielstellung zu arbeiten und andererseits können sie ihre Handlungen zur Erledigung der Aufgabe abstimmen (vgl. [Bra07]). In kompetitiven Szenarien werden durch Strategien die Konditionen einer Leistungserbringung ausgehandelt. Dabei kann mit Hilfe von auf Angebot und Nachfrage basierenden Marktmechanismen die Allokation einer Aufgabe oder einer Ressource durchgeführt werden. Marktmechanismen liegt dabei ein Protokoll zugrunde, das die genauen Regeln festlegt, nach denen die teilnehmenden Komponenten zu interagieren haben. Diese wiederum setzen individuelle Strategien ein, um das gewünschte Ergebnis zu erreichen. Eine spezielle Verhandlungsform sind dabei Auktionen (vgl. [Bra07]).

Koordination auf Ebene der Gesellschaft beinhaltet Koordinationsmechanismen, die außerhalb der Gruppe liegen und quasi von außen einwirken. Die Gesellschaft gibt dabei einen Rahmen vor, der für die enthaltenen Individuen und Gruppen verbindliche Regeln aufstellt und ihre Handlungsspielräume begrenzt. Soziale Normen und Gesetze dienen dabei dazu, das Verhalten der Individuen so einzuschränken, dass es mit den gesellschaftlichen Zielen im Einklang steht. Der Unterschied zwischen einer Norm und einem Gesetz besteht darin, dass eine Norm als eine verbindlich anerkannte Regel angesehen ist, nach der ein Individuum sein Verhalten auszurichten hat. Ein Gesetz funktioniert ähnlich als Regulativ, wird aber normalerweise im Gegensatz zur Norm in Zusammenhang mit einer Autorität betrachtet, die für seine Durchsetzung verantwortlich ist und Zuwiderhandlungen unter Strafe stellt. Das Nichteinhalten einer Norm wird im Allgemeinen nur mit sozialer Ächtung bestraft [Bra07]. Organisationsstrukturen können dazu eingesetzt werden, die zwischen Komponenten bestehenden Beziehungen festzulegen und ihre Rechte und Pflichten abzugrenzen. Dabei werden auf Basis von Konzepten wie Rollen,

Positionen und Gruppen die Handlungsmöglichkeiten der einzelnen Komponenten festgelegt. Ein hierbei häufiger verwendeter Ansatz im Bereich der Multiagentensysteme ist das *Agent-Group-Role* (AGR) Modell [FGM04].

### Selbstorganisierende Koordinationsmechanismen

Nachdem die vorangestellte Betrachtung allgemeiner Koordinationsmechanismen erste Ansatzpunkte geliefert hat, um verschiedene Mechanismen besser unterscheiden und verstehen zu können, werden im Folgenden Koordinations- und Interaktionsmechanismen betrachtet, deren Fokus auf selbstorganisierenden Systemen liegt. Dazu wird die Arbeit von *Mamei et al.* [MMTZ06] herangezogen, in der verschiedene naturinspirierte Koordinationsmechanismen untersucht und eingeteilt wurden. Die Autoren fokussieren sich dabei auf Koordinationsmechanismen, die auf lose gekoppelter, d.h. indirekter Interaktion beruhen. Abbildung 2.8 gibt diese Klassifizierung wieder und zeigt die Einteilung anhand dreier Dimension mit Bezug auf die zugrundeliegenden Charakteristika der Interaktion. Durch die Dimensionen wird dabei unterschieden, wie sich Informationen ausbreiten (1), wie diese dabei genutzt werden (2) und welche Art von Informationen überhaupt kommuniziert werden (3).

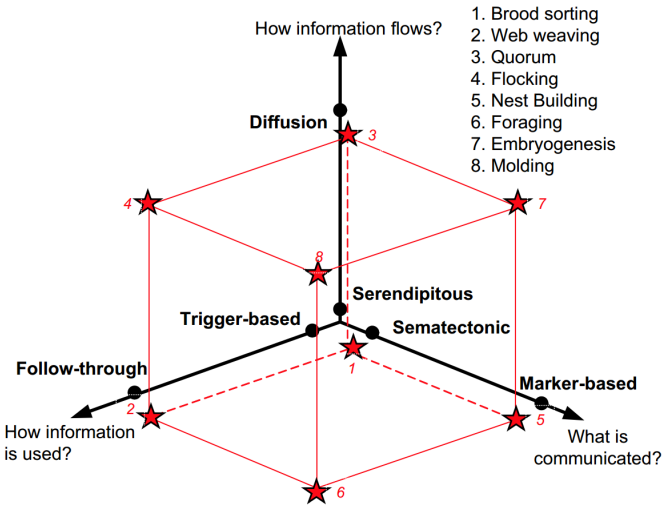



**Abbildung 2.8.:** Interaktionsmechanismen und assoziierte Selbstorganisationsmetaphern - aus (MMTZ06, S. 17).

Neben der Aufstellung des Klassifizierungsschemas identifizieren die Autoren von [MMTZ06] konkrete naturinspirierte Koordinationsmechanismen und ordnen diese dem Schema zu (vgl. Abbildung 2.8). Im Kontext von selbstorganisierenden Systemen wird z.B. häufig von Schwarmverhalten gesprochen. In [MMTZ06] wird der zugrundeliegende Koordinationsmechanismus als *„Flocking“* bezeichnet. Ein weiterer häufig verwendeter Koordinationsmechanismus, der meistens im Kontext von Ameisenalgorithmen genannt wird, findet sich als *„Foraging“* in der Abbildung wieder. Die Identifizierung und

Charakterisierung dezentraler Koordinationsmuster, wie sie in [MMTZ06] präsentiert wird, ist von essentieller Bedeutung, um deren Funktionsweise zu verstehen und ihre Anwendung auf technische Systeme zu ermöglichen. Daher finden sich in [MMTZ06] auch zahlreiche weitere Beispiele für naturinspirierte Koordinationsmechanismen, die als Inspirationsquelle dienen können, um aktuelle Probleme und Herausforderungen innerhalb verteilter Systeme zu lösen.

Im Gegensatz zur der bisher erfolgten Identifizierung und Kategorisierung von Unterscheidungsmerkmalen von Koordinationsmechanismen wird in [WH06] ein problemorientierter Ansatz beschrieben, der einen konstruktiven Zugang zu Koordinationsmechanismen ermöglichen soll. Dahinter steht das Ziel der Autoren, Entwurfsmuster zu identifizieren, wie sie beispielsweise in der Softwareentwicklung verwendet werden, sodass verteile Koordinationsmechanismen systematisch in selbstorganisierenden Systemen eingesetzt werden können. Dabei werden in [WH06] fünf Entwurfsmuster identifiziert. Tabelle 2.2 gibt in verkürzter Form verschiedene Probleme wieder, die dezentrale Koordination erfordern und nennt spezielle Koordinationsmechanismen, die zur Lösung des Problems beitragen können.

**Tabelle 2.2.:** Problemorientiertes und musterbasiertes Verständnis von Koordinationsmechanismen - nach (WH06).

| **Problemklasse** | **Koordinationsmechanismus** |
|---|---|
| Wege-Routing;<br>Informationsintegration | Digitale Pheromone (z.B. [VDPBS02]) |
| Koordination räumlicher Bewegung;<br>Bildung von Strukturen und Formationen;<br>Integration von Kontextinformationen | Gradientenfelder (z.B. [MZL04]) |
| Allokation von Ressourcen;<br>Broker-Funktionalität | Marktbasierte Koordination (z.B. [VHL+11]) |
| Vertrauen und Reputation;<br>Gruppenbildung | Tags (z.B. [Hol93]) |
| Zugriffskontrolle;<br>Rollen-Allokation;<br>Informationsaustausch | Tokens (z.B. [XSY+05]) |

Die in diesem Abschnitt aufgeführten dezentralen Koordinationsmechanismen geben im Wesentlichen die Konstruktionsmuster wieder, die in selbstorganisierenden Systemen verwendet werden, um die lokalen Interaktionen zwischen den Systemkomponenten zielorientiert zu steuern (siehe dazu auch [SR10b]). Bei einer genaueren Betrachtung dieser Koordinationsmechanismen fällt auf, dass diese zumeist auf indirekter Interaktion basieren, die wiederum häufig kommunikationslos funktioniert.

## 2.4. Selbstorganisierende Eigenschaften in verteilten Systemen

Selbstorganisierende Systeme verfügen über eine Reihe von charakteristischen Eigenschaften, die das Konzept für die Entwicklung moderner verteilter Systeme, die in komplexen, dynamischen und schwer vorhersehbaren Umgebungen agieren müssen, interessant machen. Tabelle 2.3 zeigt dabei die wesentlichen Charakteristika, basierend auf der in der Fachliteratur verwendeten Terminologie. *„Graceful degradation"* bezeichnet dabei eine auf Stabilität ausgerichtete Reaktion eines Computersystems auf Fehlerfälle bzw. unerwartete Ereignisse, bei der versucht wird, den Betrieb soweit wie möglich aufrecht zu erhalten. Ein Fehler in einem Teilsystem reduziert somit die Funktionalität des Gesamtsystems nur teilweise.

**Tabelle 2.3.:** Wesentliche Charakteristika selbstorganisierender Systeme - basierend auf (Vil13, S. 34) und erweitert um zusätzliche Referenzen.

| Eigenschaft | Nachweis in Literatur |
|---|---|
| Adaptivität | [DWH05, DMSGK05, PB05, HWMH06, SBP+09, VCO09, PR12, PRV13, Pro13, DPR15a] |
| Dezentrale Kontrolle | [DWH05, DMSGK05, PB05, HWMH06, SBP+09, DMSFR10, PR12, PRV13] |
| Flexibilität | [PB05, DPR15a] |
| Robustheit | [DMSGK05, PB05, HWMH06, SBP+09, VCO09, DMSFR10, PR12, Pro13] |
| Skalierbarkeit | [PB05, SBP+09, PR12, PRV13, Pro13] |
| Zuverlässigkeit/Fehlertoleranz | [PB05, SBP+09, DMSFR10, PR12, PRV13, PDR15a] |
| Graceful degradation | [Pro13] |
| Dynamische Strukturbildung | [DMSGK05, Pro13] |
| Multistabilität | [DMSGK05, HWMH06] |
| Einfach Systembausteine | [DMSFR10] |
| Redundanz | [DMSGK05, DMSFR10] |
| Lokalität | [DMSGK05, HWMH06, VCO09, DMSFR10] |

Allgemein lässt sich feststellen, dass selbstorganisierende Systeme sich auf ihrer mikroskopischen Ebene durch eine relative Schlichtheit auszeichnen. Dabei besitzen die einzelnen Bestandteile eines solchen Systems nur eine lokale Sicht, interagieren dementsprechend hauptsächlich mit ihren lokalen Nachbarkomponenten und zeichnen sich i.d.R. durch eine einfache Verhaltensweise aus. Nichtsdestotrotz erlangt das Gesamtsystem in einem selbstorganisierenden System eine vorgesehene Struktur und erbringt eine bestimmte Funktionalität. Damit offenbaren sich auf der, letztendlich entscheidenden, makroskopischen Ebene Eigenschaften selbstorganisierender Systeme, welche die Relevanz des Konzepts für verteilte Systeme verdeutlicht. Aufgrund ihrer Adaptivität sind selbstorganisierende Systeme dazu in der Lage, sich immer wieder auf neue Änderungen in der Umgebung anzupassen. Dabei konvergieren solche Systeme idealerweise, um so wünschenswerte Systemstrukturen herausbilden und nachteilige zu vermeiden. Durch den dezentralen Ansatz, der selbstorganisierenden Systemen zugrunde liegt, sind diese robust gegen Ausfälle und Schäden, da sie keine Komponente besitzen, deren einzelner Ausfall das Gesamtsystem gefährdet. Das bedeutet auch, dass sich die Funktionalität in Fehlersituationen nicht komplett, sondern schrittweise

reduziert. Außerdem skalieren selbstorganisierende Systeme aufgrund ihrer dezentral- und lokal-orientierten Funktionsweise sehr gut.

Als Konsequenz aus den betrachteten charakteristischen Eigenschaften selbstorganisierender Systeme lassen sich nach *Zambonelli et. al* [ZVDP03] vier Merkmale nennen, die traditionelle verteile Systeme von modernen verteilten Systemen unterscheiden:

1. **Situationsgebundenheit:** Der Kontext bzw. die Umgebung von verteilen Systemen müssen bei deren Ausführung explizit mit eingebunden werden, da das verteilte System den Kontext beeinflussen und wiederum selber von diesem beeinflusst werden kann. Daher ist eine isolierte Betrachtung des Systems von seinem Kontext bzw. seiner Umgebung nicht möglich.

2. **Offenheit:** Dadurch, dass verteilte Systeme mit anderen externen Systemen interagieren, lassen sich die Grenzen von verteilten Systemen nicht mehr eindeutig definieren. Ein verteiltes System kann so aus einer übergeordneten Perspektive als Teilsystem erscheinen. Außerdem müssen zukünftige Interaktionspartner nicht zwangsweise zur Entwicklungszeit bekannt sein und können sich erst zur Laufzeit herausbilden. Daher sind offene und standardisierte Protokolle erforderlich sowie die Fähigkeit von verteilten Systemen, sich dynamisch anzupassen.

3. **Lokale Kontrolle:** Die zunehmende Größe von verteilten Systeme, die mit der gleichzeitigen Anforderung, dynamisch agieren zu können, einher geht, lässt sich nur schwer durch globale bzw. zentrale Kontrolle erreichen. Daher ist die Eigenschaft der Autonomie gefragt, sodass verteilte Systeme aus autonomen Komponenten bestehen, die eigenständig und proaktiv handeln können, um lokale Kontrolle zu erreichen.

4. **Lokale Interaktion:** Die eben erwähnte Anforderung lokaler Kontrolle bedeutet auch, dass die autonomen Komponenten in der Lage sind, lokal zu interagieren.

Diese Merkmale sorgen für eine quantitative Steigerung der Interkonnektivität in Computersystemen [ZVDP03] und sind ein weiterer Grund für die bereits erwähnte Software-Komplexitätskrise (vgl. [KC03]).

Anhand der Diskussion der charakteristischen Eigenschaften von selbstorganisierenden Systeme wird deutlich, dass bei dieser Art von Systemen die eigentliche Performanz der Kernfunktionalität nicht im Vordergrund steht (vgl. [Vil13]). Vielmehr lassen sich die beschriebenen Eigenschaften auf die Erfüllung nichtfunktionaler Anforderungen moderner verteilter Systeme abbilden. Grundsätzlich gibt es unterschiedliche Ansätze zur Klassifikation von Anforderungen. Am verbreitetsten ist die Unterteilung in *funktionale* und *nichtfunktionale*. Während erstere festlegen, was ein Produkt, z.B. ein Softwaresystem, tun soll, beschreiben die nichtfunktionalen Eigenschaften, wie gut das System die Leistungen erbringend soll [CNYM99]. Daher werden sie

vielfach auch als Randbedingungen oder Qualitätsmerkmale verstanden. Die Bedeutung nichtfunktionaler Anforderungen im Hinblick auf die Entwicklung verteilter Systeme wird durch Norm 9126 der *International Organization for Standardization* (ISO) beschrieben (siehe Abbildung 2.9). In der Norm wird ein Modell für Softwarequalität beschrieben, in dem die Kernfunktionalität nur eine von sechs Säulen ist, nach der die Qualität einer Software bestimmt ist. Viele der anderen Säulen zielen auf die nichtfunktionalen Anforderungen ab und belegen damit deren große Bedeutung. Außerdem wird deutlich, welches hohes Maß an Übereinstimmung es zwischen den diskutierten Eigenschaften selbstorganisierender Systeme gibt.

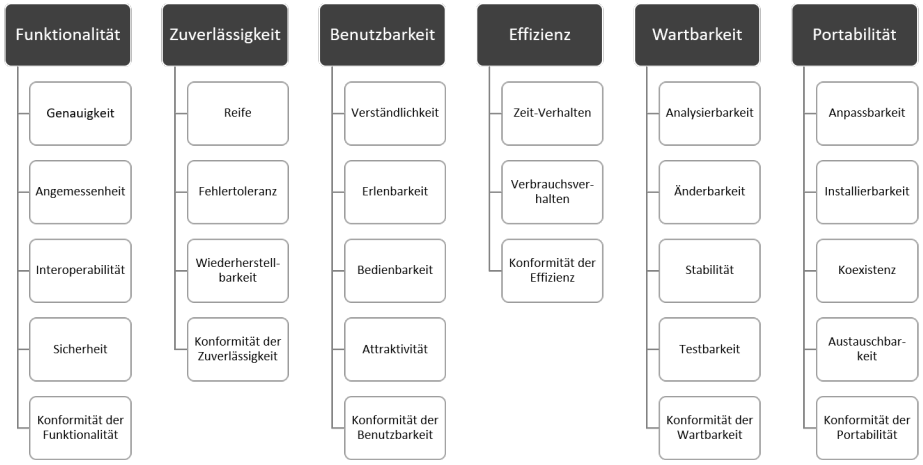



**Abbildung 2.9.:** ISO 9126 Modell für Softwarequalität.

Die wachsende Bedeutung nichtfunktionaler Anforderungen für moderne verteilte Systeme ergibt sich nicht zuletzt aus deren zunehmender Verbreitung und Durchsetzung. So werden heutzutage wichtige Nervenzentren unserer Gesellschaft, wie Produktions-, Verkehrs und Elektrizitätssysteme, maßgeblich von verteilten Systemen gesteuert. Hier wird deutlich, wie wichtig nichtfunktionale Eigenschaften wie Robustheit, Zuverlässigkeit und Skalierbarkeit sind. Im folgenden Abschnitt werden daher drei Beispiele für selbstorganisierende Systeme genannt, welche die Bereiche Produktion, Elektrizität und Verkehr (Mobilität) abdecken.

## 2.5. Anwendung selbstorganisierender Systeme

In der gängigen Fachliteratur gibt es viele verschiedene Beispiele für selbstorganisierende Systeme (vgl. Tabelle 2.3), die sich über unterschiedliche Anwendungsdomänen erstrecken. Im Folgenden werden exemplarisch drei Beispiele beschrieben, in denen Anwendungen aus unterschiedlichen Bereichen von selbstorganisierenden Eigenschaften profitieren.

### 2.5.1. Selbstheilendes Ressourcenflusssystem

Selbstheilende Systeme sind eine Unterklasse der selbstadaptiven Systeme, die in der Lage sind, eigenständig auf Fehlerfälle zu reagieren und ihr Verhalten so adaptieren können, dass sie den Fehlerfall eigenständig überwinden können. Selbstorganisierende Ressourcenflusssysteme sind wiederum eine spezialisierte Ausprägung dieser Unterklasse. Solche Systeme finden z.B. im Bereich der Web-Service Orchestrierung oder in der Produktions- und Transport-Logistik Anwendung. Ein Beispiel für ein selbstheilendes Ressourcenflusssystem ist in Abbildung 2.10 dargestellt. Dabei bearbeiten autonome Roboter Teile eines Autos, die von sogenannten *„autonomous guided vehicles“* (AGV), also selbstständig fahrenden Robotern, zwischen den einzelnen Robotern hin und her transportiert werden. Der erste Roboter (A) schweißt den Rahmen zusammen, der zweite Roboter (B) setzt die Elektronik ein, der dritte Roboter (C) baut den Motor ein, der vierte Roboter (D) befestigt den Auspuff und der fünfte Roboter (E) schraubt schließlich die Reifen fest. Zwischen den einzelnen Stationen wird das Werkstück, in diesem Fall das Auto, dabei mit Hilfe der AGV transportiert. Die Abbildung zeigt weiterhin, dass die Roboter mehrere Fähigkeiten haben, die sie anwenden können, von denen aktuell nur die hervorgehobene angewendet wird. Der Selbstheilungsmechanismus basiert hierbei auf dieser redundanten Verfügbarkeit der Fertigungsfähigkeiten der Roboter. Wenn aufgrund eines Fehlerfalls ein Roboter nicht mehr in der Lage ist, seine ursprüngliche Fähigkeit anzuwenden, kann der Fertigungsprozess entsprechend umkonfiguriert werden, sodass ein anderer Roboter die Aufgabe übernehmen kann. Dazu muss im System allerdings eine ausreichende Redundanz an Fähigkeiten vorhanden sein, sodass nach der Rekonfiguration noch immer jede Aufgabe von einem Roboter übernommen werden kann und keiner der Roboter mehr als eine Aufgabe übernehmen muss.

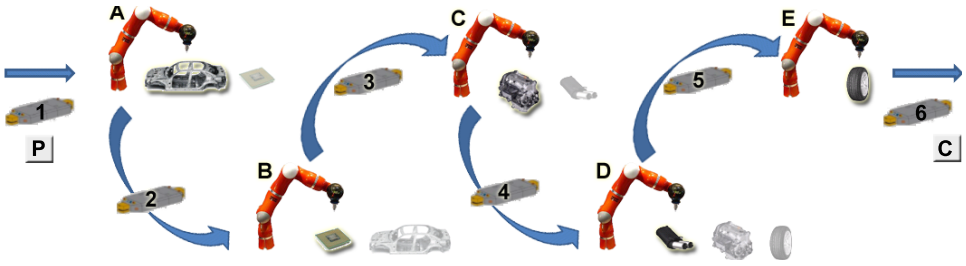



**Abbildung 2.10.:** Fertigungsprozess eines Autos in einem Ressourcenflusssystem bestehend aus Robotern mit unterschiedlichen Fähigkeiten - aus (SSS+12).

Grundsätzlich gibt zwei Lösungsmöglichkeiten um diese Rekonfiguration durchzuführen: Im Rahmen von [HNS+11] wurde ein zentraler Lösungsansatz untersucht, der auf dem *Organic Design Pattern* [GNO+08], einer Ausprägung des *Organic Computing* (siehe Abschnitt 2.2.4), basiert. Hierbei wird das Ressourcenflusssystem im Falle eines Ausfalls der Fähigkeit eines der Roboter angehalten und von einer zentralen Lösungseinheit[3] eine optimale Rekon-

[3] In diesem Fall wird ein sogenannter *Constraint-Solver* eingesetzt.

figuration der Roboter bestimmt. Basierend auf diesem Ergebnis erfolgt die Rekonfiguration und anschließend wird das System wieder gestartet. Alternativ wurde in [SSS+10a, SSS+10b, SSS+12] eine Lösung beschrieben, die den Selbstheilungsmechanismus auf Basis von dezentralen Koordinationsprozessen (siehe Abschnitt 2.3) realisiert.

Zwei wichtige Aspekte bei der Entwicklung von selbstorganisierenden Koordinationsstrategien für selbstheilende Systeme, die berücksichtigt werden sollten, sind *Skalierbarkeit* und *Robustheit*. Die entwickelte Lösung sollte dabei robust in dem Sinne sein, dass die Funktionalität des Systems nach einem Fehlerfall wiederhergestellt werden kann. Gleichzeitig sollte der Lösungsansatz gut skalieren. Das bedeutet, dass die Ressourcen, die für den Rekonfigurationsprozess benötigt werden, im schlimmsten Fall linear mit der Größe des Systems ansteigen dürfen. Im Fall der zentralen Lösung basierend auf einem *Constraint Solver* (siehe [HNS+11] kann die Robustheit des Ansatzes untersucht werden, indem validiert wird, dass dieser immer eine Lösung findet, sofern es eine mögliche Lösung gibt. Allerdings skalieren solche *Constraint-Solver* i.d.R. nicht gut, das bedeutet, dass die Rechenzeit exponentiell mit der Größe des Systems zunimmt. Dem Vorteil der Robustheit (und deren relativ einfacher Nachweisbarkeit) steht also der Nachteil der schlechten Skalierbarkeit gegenüber. Dezentrale Lösungen skalieren i.d.R. besser, sind dafür aber nicht immer in der Lage die (optimale) Lösung zu finden, da ihnen globales Wissen über den Lösungsraum fehlt, sodass Entscheidungen nur auf Basis von lokalen Wissen getroffen werden können. Eine Untersuchung der Robustheit und Skalierbarkeit des in [SSS+10a, SSS+10b, SSS+12] beschriebenen dezentralen Lösungsansatz erfolgte in [PR12]. Im Folgenden wird anhand der in [PR12] beschriebenen Rekonfigurationsstrategie aufgezeigt, wie selbstheilendes Verhalten auf Basis von dezentraler Koordination realisiert werden kann.

Der dezentrale Rekonfigurationsansatz basiert auf der bereits erwähnten Annahme, dass die Roboter mehrere, unterschiedliche Fähigkeiten (engl. *„capabilities“)* auf die zu bearbeitenden Ressourcen anwenden können. Der Rekonfigurationsprozess wird daher angestoßen, wenn einer oder mehrere Roboter nicht mehr in der Lage sind, ihre Aufgabe im Fertigungsprozess zu übernehmen, da sie aufgrund eines Defektes eine oder mehrere ihrer Fähigkeiten nicht mehr anwenden können. In diesem Fall wäre das gesamte System nicht mehr in der Lage, seine Aufgabe zu erfüllen, sofern keine Selbstheilung erfolgt. Basierend auf der Annahme, dass die Fähigkeiten der Roboter redundant sind, kann der Ressourcenfluss wiederhergestellt werden, indem Roboter ihre Aufgaben dabei entsprechend tauschen. Das bedeutet, dass Roboter die Aufgabe abgeben, die sie aufgrund ihrer defekten Fähigkeit nicht länger ausüben können und dafür von einem anderen Roboter dessen bisherige Aufgabe übernehmen, sofern sie die dafür benötigte Fähigkeit besitzen. Entsprechend übernimmt der andere Roboter diejenige Aufgabe, welche der defekte Roboter nicht länger ausüben kann. Dieser einfache Fall wird in diesem Kontext als *direkter* Tausch der Aufgaben zwischen zwei Robotern

bezeichnet. Allerdings ist nicht immer gewährleistet, dass zwei Roboter direkt ihre Aufgaben tauschen können; in diesem Fall kann es nötig sein, dass mehrere Roboter ihre Aufgaben untereinander in einem sogenannten *transitiven* Tausch austauschen, bis sich wieder eine Konfiguration ergibt, in der alle Aufgaben übernommen werden können. Ein solcher transitiver Tausch lässt sich auf Basis einer zentralen Lösung, z.B. mit einem *Constraint Solver* recht einfach bestimmen. Wenn allerdings eine dezentrale Lösung realisiert werden soll, bei dem keine Komponente im System über globalen Wissen darüber verfügt, welche Fähigkeiten alle Roboter besitzen, erweist sich die Lösung des Problems als schwieriger. Abbildung 2.11 skizziert den in [SSS+10a, SSS+10b, SSS+12] entworfenen dezentralen Lösungsansatz, der diese Schwierigkeit überwinden kann:

1. Das dargestellte Ressourcenflusssystem besteht aus drei Robotern, zwischen denen zwei AGV (Cart) die produzierten Werkstücke hin und her transportieren. Zusätzlich gibt es mit dem *Producer* und dem *Consumer* zwei spezielle Rollen, welche die zu bearbeitenden Rohlinge in das System einspeisen bzw. die fertig bearbeitenden Werkstücke einlagern. Jeder Roboter verfügt über eine Reihe von Fähigkeiten, die er ausüben kann. Die oberste ist dabei diejenige, die er aktuell anwendet.

2. Aufgrund eines Fehlers in *Robot 1* ist dieser nun nicht mehr in der Lage, *Cap. 1* auszuführen. Daher schickt er einen sogenannten *Help Request*, also eine Hilfsanfrage, entlang eines vordefinierten Kommunikationsgraphen, der dem Ressourcenfluss entspricht. In der Abbildung nicht dargestellt ist eine zusätzliche Verbindung zwischen dem *Producer* und den *Consumer*, die den Kommunikationsgraphen zu einem Token-Ring verbindet. Der *Help Request* wird solange weitergeleitet, bis er *Robot 2* erreicht, der in der Lage ist die benötigte Fähigkeiten anzuwenden.

3. Die Antwort (*Help Reply*) wird nun entgegengesetzt des Ressourcenflusses zu *Robot 1* und allen betroffenen Carts, die daraufhin ihre Routen ändern müssen, geschickt. Die betroffenen Roboter und Carts passen ihre lokale Konfiguration entsprechend an, sodass ein korrekter Ressourcenfluss wieder gewährleistet ist.

4. Da *Robot 2* zwar in der Lage war, *Cap. 1* für *Robot 1* zu übernehmen, *Robot 1* aber nicht in der Lage ist, dafür im Tausch *Cap. 2* für *Robot 2* auszuführen, bleibt *Robot 1* nach dem Tausch noch immer in einem problematischen Zustand und benötigt weitere Hilfe. Daher sendet er erneut einen *Help Request* entlang des Ressourcenflusses, dieses Mal für *Cap. 2*.

5. Diese Anfrage erreicht nun *Robot 3*, der in der Lage ist, die benötigte *Cap. 2* auszuführen. Entsprechend werden wieder die Antworten verschickt und die betroffenen Roboter und AVGs konfigurieren sich entsprechend um.

6. Da *Robot 1* in der Lage ist, im direkten Tausch die zuvor von *Robot 3* ausgeführte *Cap. 3* zu übernehmen, befindet sich das System nun wieder in einem funktionierenden Zustand.

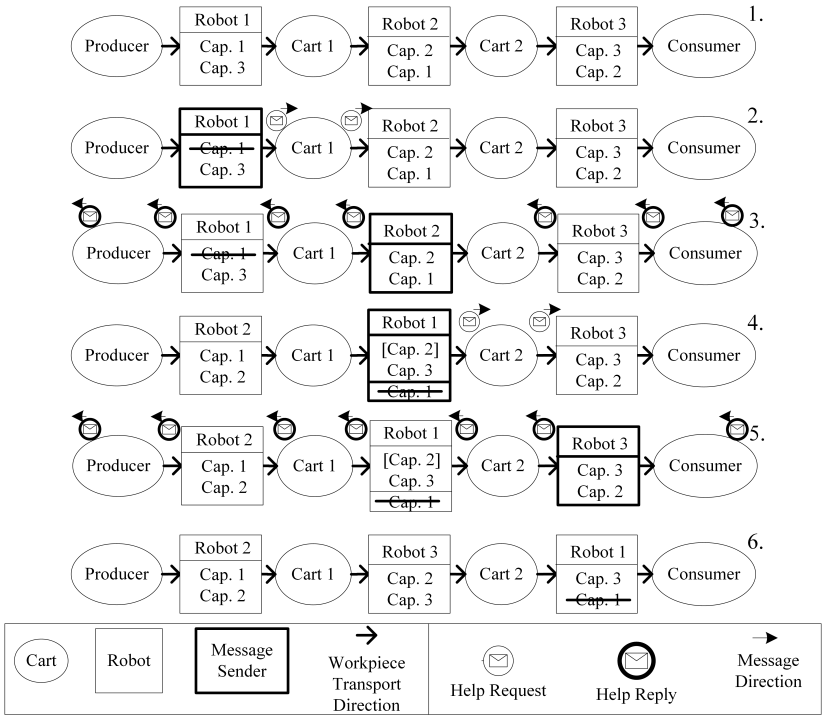



**Abbildung 2.11.:** Beispielhafte Darstellung eines dezentralen Rekonfigurationsprozesses für ein Ressourcenflusssystem - nach (PR12).

Die Skalierbarkeit des vorgestellten dezentralen Rekonfigurationsansatzes wurde im Rahmen von [PR12] untersucht. Neben einer Multiagenten-basierten Simulation wurde die Strategie auf ein stochastisches Modell abgebildet. Dazu wurde zunächst die Korrektheit des stochastischen Modells gegen die Multiagentensimulation validiert. Da dabei keine systematischen Abweichungen zwischen dem stochastischen Modell und dem implementierten Multiagentensystem festgestellt werden konnte, konnte das Modell anschließend verwendet werden, um aufzuzeigen, dass die entwickelte, dezentrale Lösung auch dann noch gut skaliert, wenn die Größe des Systems praktisch die Rechenleistung aktueller Hardware übersteigt. In [PR12] wurde ebenfalls aufgezeigt, dass die vorgestellte dezentrale Rekonfigurationsstrategie robust ist, in dem Sinne, dass bei einer ausreichenden Anzahl von Robotern[4] das System mit über 99% Wahrscheinlichkeit eine Lösung findet, sofern eine existiert.

[4]Die Untersuchungen im Rahmen von [PR12] haben dabei eine benötigte Anzahl von 40 Robotern identifiziert.

### 2.5.2. Selbstorganisierendes Fahrradverleihsystem

Aufgrund von Klimaveränderungen und schwindenden Reserven an fossilen Brennstoffen sowie zunehmender Lärmbelästigung und Verkehrsbelastung stehen Autos zunehmend in der Diskussion, wenn es um den Individualverkehr innerhalb von Städten geht. Das hat dazu geführt, dass Fahrräder als innerstädtische Verkehrsmittel zunehmend an Bedeutung gewinnen. Dies wird zusätzlich durch ein zunehmendes Gesundheits- und Umweltbewusstsein in der Bevölkerung unterstützt. Fahrräder bieten außerdem den Vorteil, dass es möglich ist, Teile einer Stadt zu erreichen, die keine direkte Anbindung an öffentliche Verkehrsmittel haben. All diese Gründe haben dazu beigetragen, dass die weltweite Verbreitung von Fahrradverleih-Systemen in den letzten Jahren rapide zugenommen hat (vgl. [VM10]). Ein weiterer Faktor, der dieses begünstigt hat, ist die Tatsache, das jüngere Generationen und Teile der urbanen Bevölkerung Autos nicht länger als wichtige Statussymbole ansehen [CK09]. Natürlich haben Fahrräder im Vergleich zu anderen Transportmitteln auch Nachteile, da ihre Verwendung z.B. stark von Wetterbedingungen und der örtlichen Topografie abhängen. Daher erscheinen Fahrräder auch eher für das Zurücklegen kurzer Wegstrecken geeignet [DG04]. Deswegen werden moderne Fahrradverleihsysteme auch hauptsächlich für kurzzeitige Ausleihen verwendet und dienen i.d.R. als Erweiterung eines bereits vorhandenen öffentlichen Verkehrsnetzes. Ein entscheidender Faktor für den wachsenden Erfolg von Fahrradverleih-Systemen ist die Einführung von Informationssystemen, die den gesamten Ausleihvorgang unterstützen [Büh08]. Viele Städte weltweit zielen darauf ab, Fahrradverleih-Systeme einzuführen, um die Luftqualität zu verbessern und Verkehrsbehinderungen in Form von Staus zu reduzieren [Mid09].

Der Betrieb von modernen Fahrradverleihsystemen in großen Städten hängt hauptsächlich von der *Verfügbarkeit* von Fahrrädern an den Ausleihstationen ab. Allerdings sind die Systeme dabei mit dem Problem der *Zuverlässigkeit* konfrontiert. Gerade Berufsverkehr-Situationen sorgen dafür, dass einige Stationen leer- und andere wiederum volllaufen. Das führt zu unzufriedenen Kunden, wenn diese keine Fahrräder an den gewünschten Stationen ausleihen bzw. zurückgeben können. Daher ist die Planung und Umsetzung der Umverteilung von Fahrrädern essentiell, um die Zuverlässigkeit des Systems zu gewährleisten. Dabei gibt es sowohl in der Forschung als auch in der Praxis zahlreiche unterschiedliche Ansätze, um das Problem der Zuverlässigkeit zu lösen [VM10]. So analysieren *Kaltenbrunner et al.* [KMG+10] z.B. die Verfügbarkeit von Fahrräder an den Stationen für Barcelonas *Bicing* Fahrradverleih-System, um daraus zeitliche und geografische Bewegungsmuster innerhalb der Stadt abzuleiten, die dann als Grundlage dienen, um die Anzahl von verfügbaren Fahrräder an den Stationen vorherzusagen. Ein ähnlicher Ansatz wurde von *Froehlich et al.* [FNO09], ebenfalls für Barcelonas *Bicing*-System, mit der Verwendung von Clustering-Techniken verfolgt, um gemeinsame Verhaltensweisen der Benutzer zu identifizieren, mit

dem Ziel die kurzfristige Nutzung der Stationen vorhersagen zu können. Die Aktualität des Themas zeigt sich u.a. in [ZZyDB15], wo in einer empirischen Studie Fahrradverleih-Systeme in fünf chinesischen Städten verglichen wurden. Da China massiv unter den negativen Konsequenzen einer hohen Anzahl von privat genutzten Autos innerhalb großer und dicht besiedelter Städte leidet, würde es sehr von einer verstärkten Nutzung von Fahrrädern profitieren. Begünstigt wäre eine solche Entwicklung von der langen Historie der Fahrradnutzung im Land.

Da *Zuverlässigkeit* in Form der *Verfügbarkeit* von Fahrrädern an den Ausleihstationen eine der wichtigsten Herausforderung beim Betrieb von Fahrradverleihsystemen ist, erscheint es sinnvoll, Fahrradverleihsysteme mit selbstorganisierenden Verhalten auszustatten, um die gewünschte Eigenschaft zu realisieren. Dazu wird ein dieser Stelle erneut auf Tabelle 2.3 verwiesen, in der *Zuverlässigkeit* als eine charakteristische Eigenschaft von selbstorganisierenden Systemen aufgeführt ist. Dazu wurden im Rahmen dieser Dissertationen unterschiedliche selbstorganisierende Strategien mit Hinblick auf ihrem Einfluss auf die *Verfügbarkeit* von Fahrrädern an den Ausleihstationen untersucht [PRV13, PDR15a, PDR16b]. Die Untersuchungen basieren dabei auf Simulationen, die auf Basis von Washington D.C.'s *Capital Bikeshare*-System ausgeführt wurden. Das *Capital Bikeshare*-System wurde im September 2010 gestartet und war bis zum Mai 2013 das größte Fahrradverleihsystem in den Vereinigten Staaten von Amerika (vgl. [Mar10]). Aktuell verfügt das System über 345 Stationen, und im Jahre 2014 standen den Kunden zwischen 2400 und 2900 Fahrräder zur Verfügung. Alle Daten, die das System und dessen Betrieb betreffen, werden von der Betreibergesellschaft über das *Capital Bikeshare Dashboard*[5] online frei zur Verfügung gestellt. Die Simulationen basieren dabei auf Historie-Daten der durchgeführten Fahrten in 2011 und 2012 [PRV13] sowie 2014 [PDR15a, PDR16b]. Wie bei vielen andere Fahrradverleihsystemen auch, basiert das Preismodell auf einer monatlichen Gebühr und die ersten 30 Minuten einer Ausleihe sind kostenlos. Abbildung 2.12 zeigt in monatlicher Auflösung die systemweite Anzahl der durchgeführten Fahrten vom Start des Systems bis zum Ende des Jahres 2014. Dabei sind sowohl das Wachstum des Systems im Hinblick auf die Anzahl der durchgeführten Fahrten als auch der Einfluss des Wetters bzw. der Jahreszeiten auf diese zu erkennen.

Um die *Verfügbarkeit* von Fahrrädern an den Stationen sicherzustellen, verwendet die Betreibergesellschaft (Capital Bikeshare) spezielle LKWs zur Umverteilung [Mau13]. Abbildung 2.13 zeigt die monatliche Anzahl dieser Umverteilungsmaßnahmen, die von Capital Bikeshare im Jahr 2014 durchgeführt wurden. Dabei wird deutlich, dass ein Zusammenhang zwischen der Anzahl der durchgeführten Fahrten in diesem Zeitraum (vgl. Abbildung 2.12) und der Anzahl der Umverteilungsmaßnahmen besteht. Außerdem wird anhand der Abbildung deutlich, dass der Betrieb eines solchen Systems einen

[5] Capital Bikeshare Dashboard: `http://cabidashboard.ddot.dc.gov`, aufgerufen am 1. Februar 2017

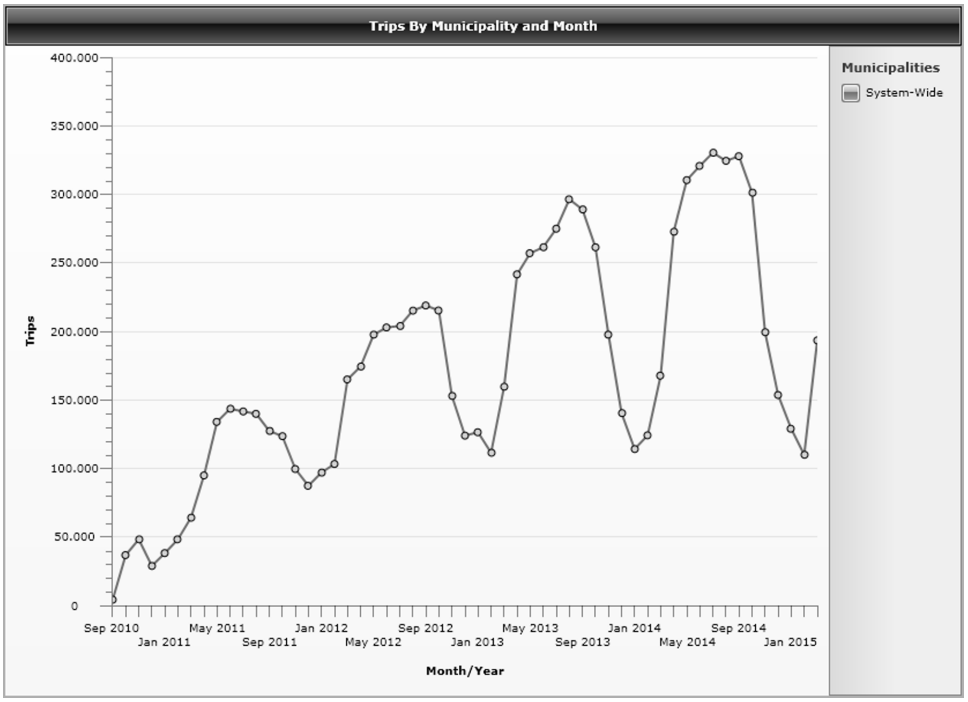



**Abbildung 2.12.:** Capital Bikeshare systemweite Anzahl von Fahrten von September 2010 bis März 2015 - aus (PDR15a, S. 8).

erheblichen Aufwand erfordert, um die *Verfügbarkeit* von Fahrrädern an den Stationen sicherzustellen.

Bei der Betrachtung von Abbildung 2.14 wird deutlich, dass trotz dieser bereits durchgeführten Umverteilung von Fahrrädern durch LKWs Stationen voll- bzw. leerlaufen. Zusätzlich zeigt die Abbildung, dass die Anzahl der leeren Stationen höher ist als die Anzahl der vollen Stationen. Dieses kann ein Indikator dafür sein, dass die Stationen so entworfen wurden, dass zusätzliche Kapazitäten (Abstellplätze für Fahrräder) eingeplant wurden, um die Chancen zu erhöhen, dass ein Nutzer ein ausgeliehenes Fahrrad an der von ihm gewünschten Station abgeben kann. Abschließend lässt sich feststellen, dass trotz der bereits durchgeführten Umverteilungsmaßnahmen weiterer Verbesserungsbedarf besteht. Dieses kann entweder durch eine zusätzliche LKW-basierte Umverteilungen von Fahrrädern geschehen oder durch die Einführung von neuen Ansätzen. Solche Ansätze, basierend auf selbstorganisierenden Strategien, wurden in [PRV13, PDR15a, PDR16b] untersucht.

Die dabei untersuchten Ansätze basieren auf einem Anreizsystem, dass die Benutzer dazu ermutigen soll, bei der Ausleihe und der Rückgabe von Fährrädern auch in der Nähe befindliche Stationen in Betracht zu ziehen. Dadurch soll eine selbstorganisierende Umverteilung der Fahrräder erreicht werden, indem die Benutzer ermutigt werden, bei der Ausleihe Stationen zu verwenden, die entweder voll sind oder über ausreichend Fahrräder verfügen und bei der Rückgabe Stationen, die entweder leer sind oder nur über wenige Fahrräder verfügen. Natürlich setzt dieser Ansatz die Bereitschaft der Be-

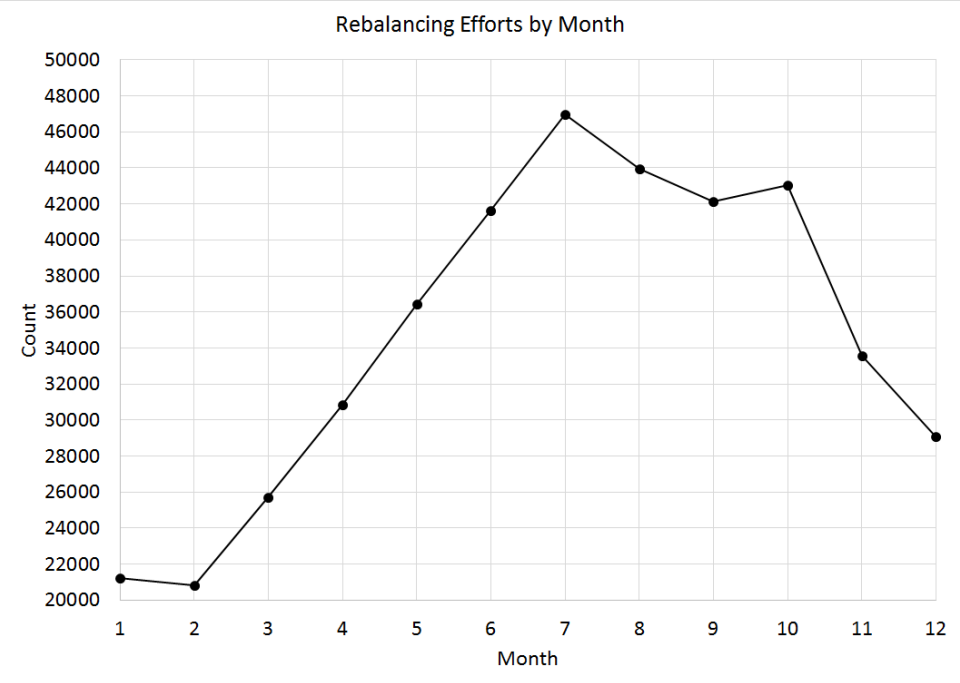



**Abbildung 2.13.:** Capital Bikeshare Umverteilungsmaßnahmen in 2014 - aus (PDR16b, S. 2).

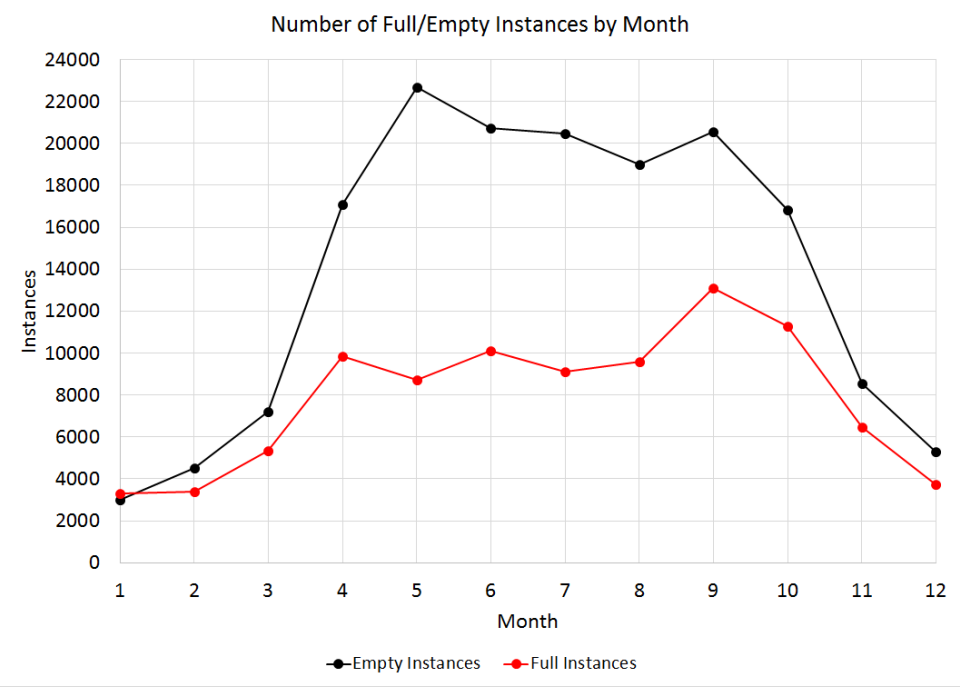



**Abbildung 2.14.:** Anzahl der leeren und vollen Stationen in 2014 - aus (PDR16b, S. 2).

nutzer voraus, Umwege in Kauf zu nehmen. Das Anreizsystem soll an dieser Stelle dafür sorgen, dass die Bereitschaft der Benutzer zu solchen Umleitungen erhöht werden, indem den Benutzern Zuwendungen in Form von Belohnungen, wie z.B. Freiminuten für die nächste Ausleihe, geboten werden. Abbildung 2.15 zeigt, wie ein solcher Ausleihvorgang aus der Perspektive eines kooperativen Benutzers aussieht. Zunächst nähert sich der Benutzer einer Ausleihstation an, um dort ein Fahrrad auszuleihen. Wenn an dieser Station keine bzw. nur noch wenige Fahrräder verfügbar sind, wird dem Benutzer eine alternative Station in der Nähe vorgeschlagen, die noch über ausreichend Fahrräder verfügt. Nun kann der Benutzer entscheiden, ob er dem Vorschlag folgt und sich zu der alternativen Ausleihstation begibt oder er, sofern an der Station noch Fahrräder verfügbar sind, an der ursprünglichen Station ein Fahrrad ausleiht. Wenn der Benutzer mit dem Fahrrad zu der Rückgabestation gefahren ist, kann ihm hier, wenn die Station voll bzw. fast voll ist, wieder eine alternative Station zur Abgabe vorgeschlagen werden. Nun kann der Benutzer erneut entscheiden, ob er dem Vorschlag folgt und zu der alternativen Rückgabestation fährt oder er das Fahrrad direkt an Ort und Stelle zurückgibt. Wie bereits erwähnt, ist neben der Bereitschaft der Benutzer, einer vorgeschlagenen Umleitung zu folgen, die Unterstützung durch ein Informationssystem für diesen Ansatz essentiell. Die Aufgabe des Informationssystems ist es dabei, die alternativen Ausleih- und Rückgabestationen zu bestimmen, die dem Benutzer vorgeschlagen werden. In [PRV13, PDR15a, PDR16b] wurden dabei unterschiedliche koordinationsunterstützte Ansätze zur Bestimmung dieser alternativen Stationen untersucht. Im Folgenden werden exemplarisch die Konzepte und Ergebnisse aus [PDR15a] präsentiert.

Der selbstorganisierende Ansatz zur Umverteilung der Fahrräder durch die Benutzer, der in [PDR15a] untersucht wurde, basiert auf einem dezentralen Koordinationsansatz zur Bestimmung der alternativen Ausleih- und Abgabestationen, die den Benutzern vorgeschlagen werden. Dabei sendet jede Station ihre aktuelle Auslastungsrate zu allen anderen Station, die sich innerhalb eines bestimmten Radius befinden. Die Auslastungsrate $\mu$ ergibt sich dabei aus der Anzahl der belegten Plätze $P_b$ geteilt durch die Anzahl der freien Plätze $P_f$, also $\mu = \frac{P_b}{P_f}$. Der Kommunikationsradius, der bestimmt, welche anderen Stationen die Koordinationsnachricht mit der Auslastungsrate erhalten, ist dabei ein variabler Parameter des Koordinationsansatzes. Stationen, die eine solche Koordinationsnachricht erhalten, benutzen diese, um dezentral die alternativen Ausleih- und Abgabestationen zu bestimmen. Dabei wird aus allen erhaltenen Nachrichten die Station mit der niedrigsten Auslastungsrate und die Station mit der höchsten Auslastungsrate bestimmt. Diese Stationen werden dann entsprechend den Benutzern als alternative Ausleih- (höchste Auslastungsrate) und Abgabestation (niedrigste Auslastungsrate) vorgeschlagen, sofern die Station selber im Falle einer Ausleihe über nur wenige bzw. keine Fahrräder verfügt bzw. im Falle einer Rückgabe bereits voll bzw. fast voll ist. Die maximale Distanz, die die Benutzer dabei als Umweg in Kauf nehmen müssen, entspricht folglich $1:1$ dem gewählten Kommunikationsradius.

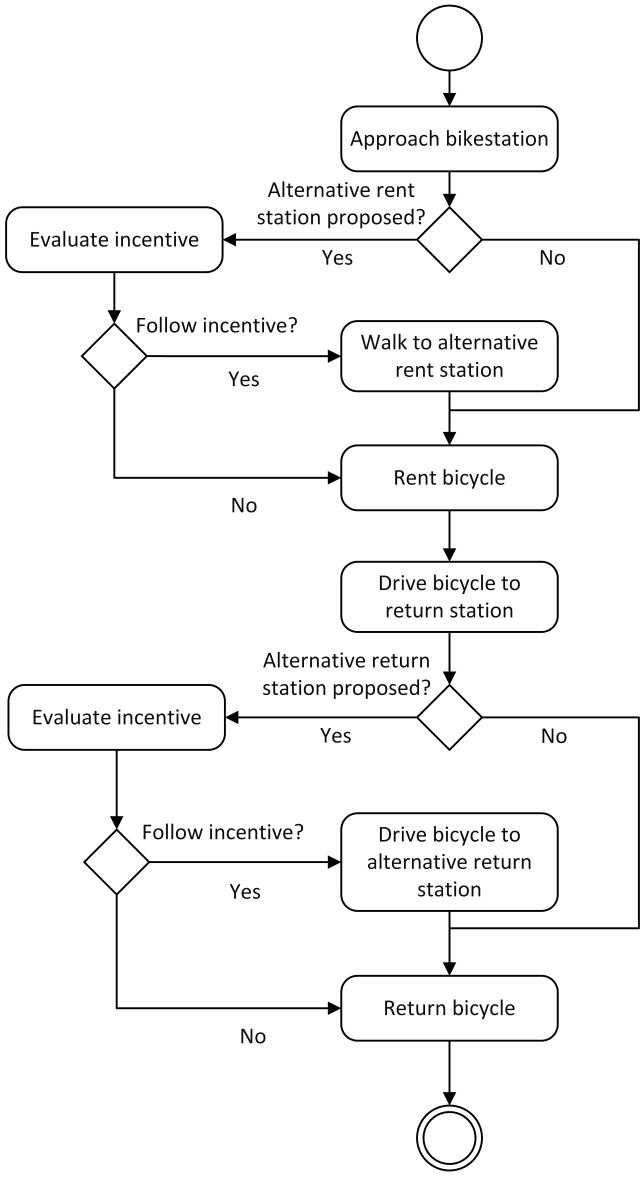



**Abbildung 2.15.:** UML Aktivitätsdiagramm das einen Ausleihvorgang mit vorgeschlagenen Alternativen aus Benutzersicht beschreibt - aus (PDR16b, S. 6).

Um den Einfluss der selbstorganisierenden Umverteilung von Fahrrädern untersuchen zu können, wurde in [PDR15a] eine Simulation auf Basis von *RinSim* [vLH12], einem Multiagenten-basierten Logistiksimulator, realisiert. Für die Simulation wurden Historie-Daten über die Ausleihvorgänge in Washington D.C. in 2014 analysiert. Dabei wurden die Daten aller Montage, mit Ausnahme von Feiertagen, betrachtet und ein idealisierter Montag auf Basis von stochastischen Wahrscheinlichkeiten erstellt. Das daraus resultierende Modell teilt den Tag in 24 Zeitintervalle, eines für jede Stunde, ein. Für jede dieser Zeitintervalle wurden basierend auf den Historie-Daten über die Ausleihvorgänge Abfahrts- und davon abhängige Ankunftswahrscheinlichkeiten für die einzelnen Stationen berechnet. Gleichung 2.1 beschreibt dabei die Abfahrtswahrscheinlichkeit $p_A$ für eine Station $A$, wobei $\phi_A$ die Anzahl der Abfahrten von dieser Station beschreibt und $\Phi$ die Gesamtanzahl der Abfahrten für die betrachtete Zeitscheibe ist. Gleichung 2.2 beschreibt die abhängige Ankunftswahrscheinlichkeit $q_A$. Diese ist charakterisiert durch den Quotient aus der Anzahl der Abfahrten von Station $A$ zu Station $B$, beschrieben als $\gamma_B$, und Gesamtanzahl der Abfahrten von Station $A$ aus, beschrieben als $\Gamma_A$.

$$p_A = \frac{\phi_A}{\Phi} \tag{2.1}$$

$$q_A = \frac{\gamma_B}{\Gamma_A} \tag{2.2}$$

Um den unterschiedlichen Andrang an den verschiedenen Stationen zu unterschiedlichen Zeiten des Tages zu simulieren, wurde der Mittelwert für jedes der 24 Zeitintervalle basierend auf dem tatsächlichen Andrang gemäß den Historie-Daten errechnet. Ausgehend von diesem Modell werden während der Ausführung der Simulation entsprechend dem mittleren Andrang für das aktuelle Zeitintervall gleichmäßig verteilt Agenten gestartet, welche die Benutzer abbilden. Basierend auf den Abfahrts- und Ankunftswahrscheinlichkeiten wird dabei bestimmt, von welcher Station zu welcher anderen Station die Agenten fahren. Die Benutzer-Agenten verhalten sich dabei entsprechend dem UML Aktivitätsdiagramm aus Abbildung 2.15. Dabei bewegen sie sich mit einer konstanten Geschwindigkeit entlang der Straßen, die auf einem Auszug aus der *OpenStreetMap* Karte [HW08] von Washington, D.C. basieren und innerhalb der Simulation als Graph abgebildet ist. Das Gewicht der Kanten repräsentiert dabei die Entfernung zum nächsten Knotenpunkt. Für die Simulation bewegen sich die Fahrradfahrer mit einer konstanten Geschwindigkeit von $18\frac{km}{h}$. Alle Stationen verfügen über $20$ Plätze, von denen initial $10$ mit Fahrrädern belegt sind. Abbildung 2.16 zeigt einen Ausschnitt der Karte dieser Simulation mit den sich überlappenden Kommunikationsradien der Ausleihstationen. Innerhalb der Simulation ist eine Überbelegung der Stationen erlaubt, d.h. ein Benutzer kann das Fahrrad auch dann an einer Station abgegeben, wenn diese eigentlich keine freien Plätze mehr hat.

Die Ergebnisse der Simulationen sind in Abbildung 2.17 dargestellt. Der obere Teil der Abbildung (a) zeigt den Verlauf der Zustände der Stationen

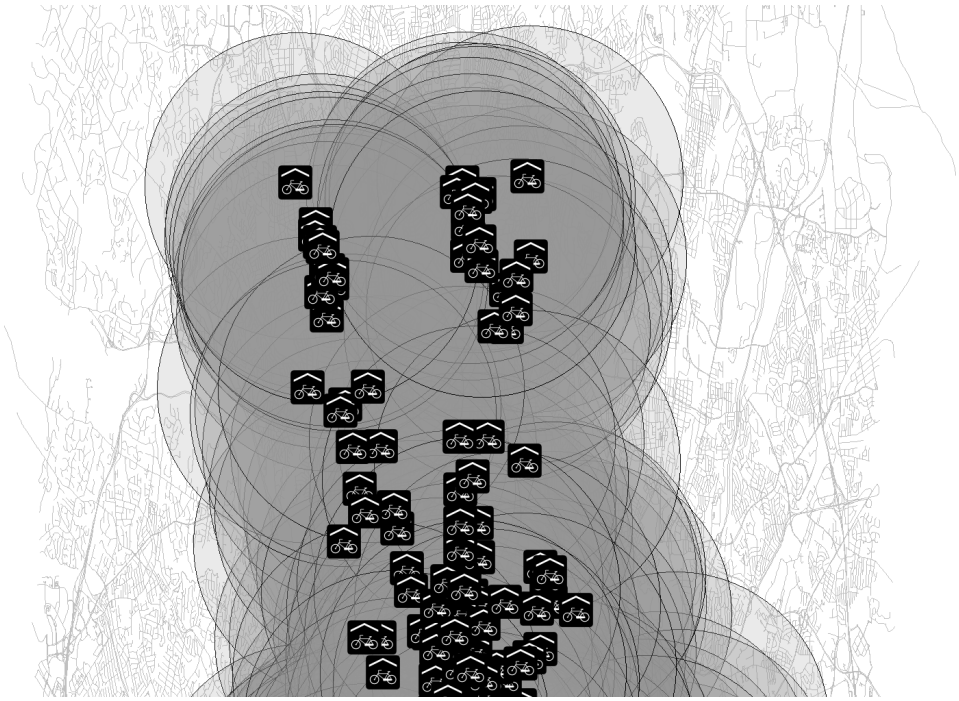
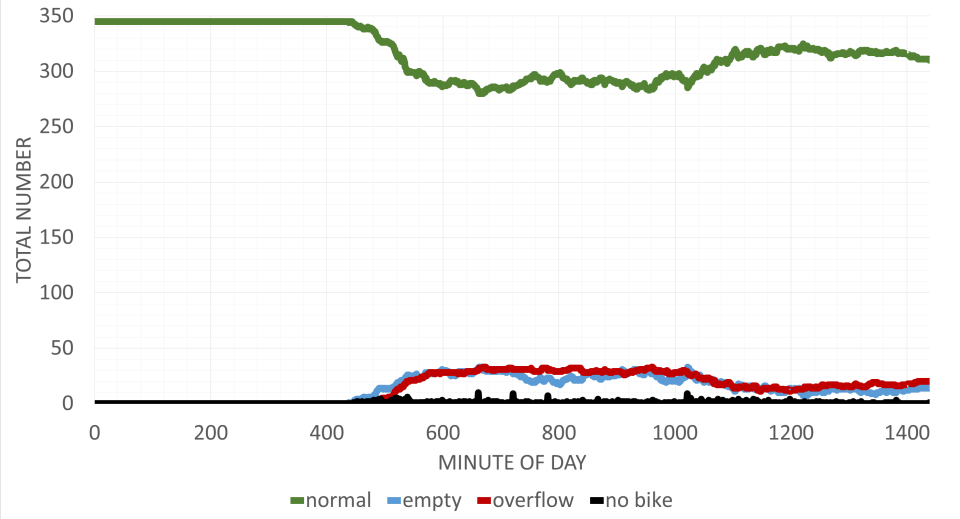


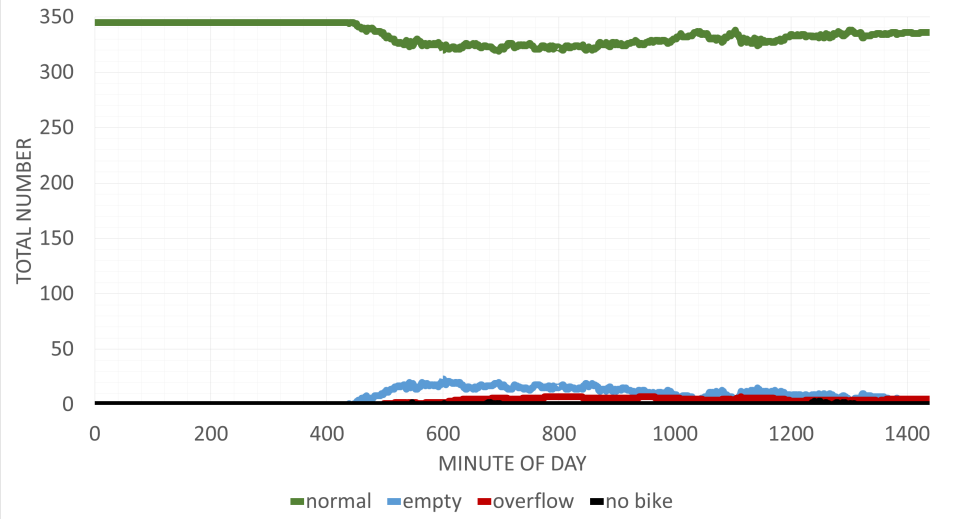
**Abbildung 2.16.:** Auszug der simulierten Karte, die das Straßenmodell und einige der Stationen mit ihrem Kommunikationsradius zeigt - aus (PDR15a, S. 19).

über den simulierten Tag für ein Referenzszenario ohne selbstorganisierende Umverteilung der Fahrräder. Die grüne Linie zeigt dabei die Anzahl an Stationen im normalen Zustand, also Stationen, die weder leer noch voll sind. Die rote Linie zeigt die Anzahl der Stationen, die voll oder überbelegt sind, die blaue Linie zeigt den Verlauf der Anzahl der leeren Stationen und die schwarze Linie gibt Auskunft über Ereignisse, bei denen eine Fahrt nicht stattfinden konnte, da an der entsprechenden Ausleihstation kein Fahrrad vorhanden war. Anhand der Abbildung lässt sich beobachten, dass die Anzahl der normalen Stationen mit dem morgendlichen Berufsverkehr (um Minute 420, was ca. 7 Uhr entspricht) abnimmt, während gleichzeitig die Anzahl der leeren und vollen/überbelegten Stationen zunimmt. Diese Werte fluktuieren zunächst nur wenig, wobei sich die Anzahl der normalen Stationen zum Nachmittag hin (etwa Minute 1000) wieder etwas erholt, während zeitgleich die Anzahl der leeren und vollen/überlegten Stationen leicht abnehmen. Dieses Verhalten lässt sich durch das Verhalten der Pendler zu den Hauptverkehrszeiten erklären. Morgens bewegen diese sich aus den Vororten in das Zentrum der Stadt und nachmittags wieder zurück. Dadurch laufen die Stationen in den Vororten morgens leer, während die Station im Stadtzentrum voll bzw. überbelegt werden. Nachmittags werden die Fahrräder dieser Stationen dann wieder entliehen und füllen kurz danach die leeren Stationen in den Vororten wieder auf.

Der untere Teil von Abbildung 2.17 (b) zeigt den Verlauf der Zustände basierend auf dem beschriebenen selbstorganisierenden Umverteilungsansatz für den Fall, dass die Benutzer immer einer vorgeschlagenen Umleitung fol-




**(a)** Verlauf der Zustände der Stationen ohne Umverteilung.




**(b)** Verlauf der Zustände der Stationen basierend auf dem dezentralen Umverteilungsansatz.

**Abbildung 2.17.:** Ergebnisse der simulationsbasierten Auswertung der Ansätze - aus (PDR15a).

gen. Dabei wurde die Bewegungsgeschwindigkeit der Benutzer auf $\frac{1}{3}$ reduziert, wenn diese noch ohne Fahrrad zu einer alternativen Ausleihstation umgeleitet werden. In diesem Szenario wurde ein Kommunikationsradius von $1km$ verwendet. Die Abbildung zeigt, dass auch in diesem Fall die Anzahl der normalen Stationen mit Beginn des morgendlichen Berufsverkehrs abnehmen, während die Anzahl der leeren und vollen bzw. überbelegten Station ansteigen. Allerdings bleibt die Anzahl der normalen Stationen hier aufgrund der Umleitung der Benutzer deutlich stabiler. Gleichzeitig ist die Anzahl der vollen bzw. überbelegten Stationen deutlich reduziert. Am signifikantesten ist die Reduzierung der in der Abbildung als „no bike“ bezeichneten Ereignisse, also der Anzahl der nicht stattfindenden Fahrten, um $90,16\%$ von $701$ auf $69$ Vorkommnisse. Die Ursache dafür, dass solche Ereignisse auch dann noch auftreten, wenn ein kooperierender Benutzer simuliert wird, ist in der Designentscheidung begründet, dass Benutzer eine vorgeschlagene Umleitung nur einmal akzeptieren. Wenn sie also schon einmal zu einer alternativen Station umgeleitet wurden, werden sie einer solchen Umleitung kein zweites Mal folgen. Daher können solche nicht stattfindenden Fahrten auftreten, wenn der Benutzer von einer leeren Ausleihstation zu einer anderen Station umgeleitet wird und diese in der Zwischenzeit ebenfalls leer läuft. Zusammenfassend lässt sich anhand der Grafik ein Eindruck darüber vermitteln, wie die selbstorganisierende Umverteilung der Fahrräder basierend auf dem dezentralen Koordinationsansatz dazu beiträgt, die *Verfügbarkeit* der Fahrräder an den Stationen zu erhöhen.

Eine weitere Untersuchung, die im Rahmen von [PDR15a] stattfand, war die Analyse des Einflusses der Kooperationsbereitschaft der Benutzer. Dafür wurde mit der *Kooperationsrate* ein Parameter eingeführt, der bestimmt, wieviel Prozent der Benutzer einer vorgeschlagenen Umleitung folgen. Dieser Wert wurde noch weiter unterteilt in die *Ausleih-Kooperationsrate* und die entsprechende *Rückgabe-Kooperationsrate*. Die Ergebnisse dieser Analyse sind in Abbildung 2.18 dargestellt. Hier wird die mittlere, relative Abweichung zwischen der Anzahl der normalen Stationen im Vergleich zum Referenzszenario ohne selbstorganisierende Umverteilung abgebildet. Der Wert gibt also Auskunft darüber, wie viel mehr Stationen im Vergleich zum Referenzszenario im Mittel über den simulierten Tag im normalen Zustand sind. Die Ergebnisse zeigen dabei eine maximale Verbesserung von etwa $9\%$. Für detaillierte Beschreibungen der hier aufgeführten Analysen sowie weitere Auswertungen der hier beschriebenen Szenarios wird an dieser Stelle auf die entsprechenden Primärquellen [PRV13, PDR15a, PDR16b] verwiesen.

### 2.5.3. Selbstoptimierendes Demand-Side-Management

Die Energieversorgung befindet sich weltweit vor einem grundlegenden Paradigmenwechsel. Der Wandel von der intensiven Nutzung fossiler Brennstoffe sowie nuklearer Energieträger hin zu einer regenerativen und nachhaltigeren Energieerzeugung basiert auf der Feststellung, dass die glob-

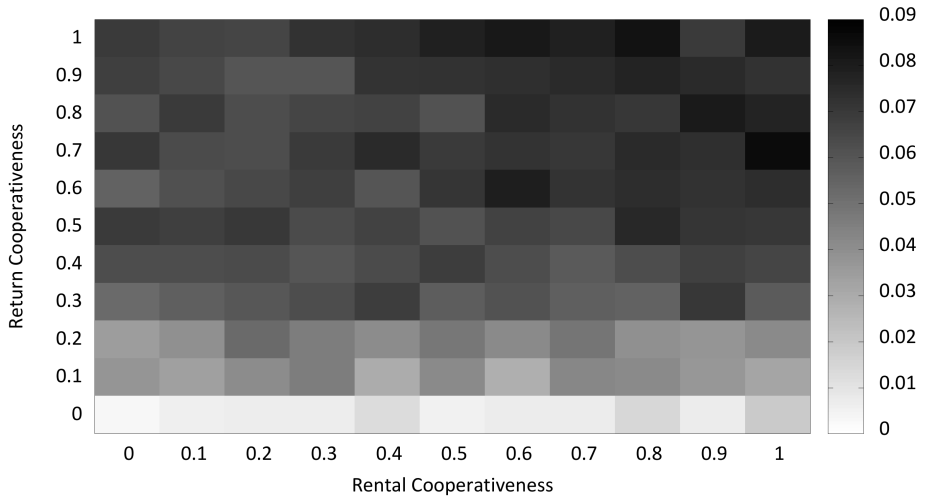



**Abbildung 2.18.:** Mittlere, relative Abweichung in der Anzahl der normalen Stationen im Vergleich zu dem Referenzszenario ohne selbstorganisierendes Verhalten - aus (PDR15a, S. 24).

ale Erderwärmung, ausgehend von der weltweit steigenden Industrialisierung, sowie der wachsenden Emissionen konventioneller Kraftwerke, die Menschen in vielen Teilen der Erde zunehmend mit klimatischen Veränderungen und extremen Wetterphänomenen konfrontiert. Auch sind die traditionellen Quellen für Energieträger wie Öl, Gas, Kohle und radioaktive Isotope begrenzt. Zwar sind bei radioaktiven Energieträgern und Kohle mittelfristig keine Engpässe abzusehen [CAR+09], doch beim Erdöl wird der kritische Zeitpunkt der maximalen globalen Förderung je nach Prognose zwischen 2020 [Ode00] und 2035 [CAR+09] erwartet. Kernenergie wurde zwar in den 1950er Jahren als billige und saubere Energieform der Zukunft gesehen. Dieser Glaube wurde jedoch durch die Kraftwerksunglücke von Tschernobyl (1986) und Fukushima (2011) weltweit stark erschüttert. Infolgedessen setzte in vielen Nationen ein Umdenken ein, so zum Beispiel durch die in Deutschland im Jahr 2011 beschlossene Energiewende[6], die den vollständigen Ausstieg aus der Kernenergie bis zum Jahr 2022 vorsieht.

Das Einspeiseziel der Bundesrepublik Deutschland und anderer Staaten für einen erhöhten Anteil erneuerbarer Energien am Strommix sowie die technische Entwicklung kleiner, preisgünstiger und rentabler Erzeuger, die mittels hochverfügbarer (Gas, Biomasse) oder gar praktisch unlimitierter Ressourcen (Sonne, Wind, Wasserkraft) betrieben werden können, führt zu einer zunehmenden Dezentralisierung und Diversifizierung der Erzeugerlage der elektrischen Energie. Die Verfügbarkeit und volle Erzeugungskapazität dieser Erzeuger hängt jedoch zum Teil von nicht beeinflussbaren und vorhersage-basierten Faktoren wie der Wetterlage ab. Dadurch geraten Energieversorgungsunternehmen in ein Spannungsfeld zwischen der vorrangigen Einspeisung regenerativer Energien durch das Erneuerbare-Energien-

[6] §7 des Dreizehnten Gesetzes zur Änderung des Atomgesetzes, Bundesgesetzblatt Nr. 43 (2011)

Gesetz (EEG)[7], der Gewährleistung der Versorgungssicherheit[8] und dem wirtschaftlichen Betrieb der eigenen Kraftwerke. Die Nutzung des Potentials der Verbraucherseite als zusätzlichen Freiheitsgrad, um die Situation abzumildern, wurde bereits in den 1980er Jahren, u.a. von [Gel85], als *Demand-Side Management* (DSM) vorgestellt. Die möglichen Anwendungsfälle des *Demand-Side Managements* sind dabei vielseitig. In der Forschung finden sich zumeist Systeme, die zum *Load Balancing* (Lastverteilung) eingesetzt werden, um den Verbrauch mit der Erzeugung abzustimmen und Abweichungen zwischen diesen auf unterschiedlichen Zeitskalen zu minimieren (siehe dazu z.B. [GMR+03, KWK05, Kam10, MCCG+12]).

Aktuelle Forschungsfragen in diesem Gebiet beschäftigen sich u.a. mit der Verbindung der Verbraucher-Seite mit dem sogenannten *Smart Grid*. Dabei sind vor allem die Koordination und die Kontrolle dieser Seite von entscheidender Bedeutung und ein kritischer Aspekt für die erfolgreiche Realisierung zukünftiger Smart Grids (vgl. [DPR15a]). Haushaltsgeräte auf der Verbraucher-Seite, wie z.B. Kühlschränke oder Waschmaschinen, haben zwar nur einen geringen Energieverbrauch, machen aber einen hohen Anteil der täglichen Lastspitzen im Energienetz aus. Daher, bieten sie ein großes Potential für DSM. Um diese Potentiale für die Lastverteilung nutzbar machen zu können, werden intelligente Kontrollkonzepte benötigt, die Energie- und Informations- und Kommunikationstechnologien miteinander verbinden. Selbstadaptive und selbstverwaltene Eigenschaften (vgl. Abschnitt 2.2) bieten hier ein vielversprechendes Konzept, um die benötigten Koordinations- und Kontrollkonzepte, die für das DSM in zukünftigen Smart Grid Anwendungen benötigt werden, adaptiv zu realisieren. Im Rahmen dieser Dissertation wurde daher in [DPR15a] ein selbstoptimierendes DSM System realisiert, das Ameisenalgorithmen im Sinne der *Ant Colony Optimization* [DB06] zur Selbstoptimierung nutzt.

Die Vorausplanungen von DSM Aktivitäten basieren meistens auf einer Abschätzung der erzeugten Energiemenge der zugehörigen verteilten Energieressourcen (engl. *„distributed energy resources“*, DER im Folgenden). Viele dieser DER hängen stark von Wetterlagen wie Wind oder Sonnenschein und dadurch direkt von der Qualität der entsprechenden Wettervorhersagen ab. Veränderungen an diesen können außerdem die Energiemärkte beeinflussen, die eigene Dynamiken und Restriktionen ausweisen. In einem Energienetz mit mehr und mehr DER müssen die entsprechenden Netzrestriktionen und -bedingungen weiterhin eingehalten werden. Ein Beleg für die steigende Bedeutung der Überwachung der entsprechenden Netzrestriktionen ist der steigende Grad der Standardisierung (z.B. IEC Standard 61850) der eingesetzten IKT-Infrastruktur. Daher lässt sich feststellen, das plötzliche Änderungen der Umwelt-, marktrelevanten- oder Netz-Bedingungen dazu führen können, dass die ursprünglichen Ziele der Energieplanung nicht eingehalten werden können. Der Einsatz entsprechender Sensoren, die Änderungen in den Be-

[7] Offizieller Titel: „Gesetz für den Vorrang Erneuerbarer Energien“, Inkrafttreten: 1. April 2000
[8] §13 Abs. 1 EnWG

dingungen der drei aufgeführten Bereiche (Umwelt, Märkte, Netz) detektieren können, würde es erlauben, mit einem entsprechend dynamischen und adaptiven DSM-System auf diese Änderungen reagieren zu können. Daher wird in [DPR15a] ein adaptiver DSM Ansatz vorgeschlagen, der es erlaubt, auf Änderungen in der Umgebung zu reagieren, indem Fahrpläne auf der Verbraucher-Seite entsprechend angepasst wird. Die Architektur, die diesem Ansatz zugrunde liegt, wird in [DBPR14] präsentiert und zeigt die Anwendung einer dezentralen Optimierung der Verbraucherseite. Das Verteilungskonzept, das der Optimierung zugrunde liegt, wird in [DPR14] präsentiert. Abbildung 2.19 zeigt die Modellierung des Problems auf Grundlage der weitverbreiteten *MAPE-K Schleife* aus dem *Autonomic Computing* Ansatz [KC03]. Abweichungen, z.B. aufgrund von sich verändernden Wettervorhersagen, werden überwacht und analysiert. Wenn durch diese externen Einflüsse signifikante Auswirkungen auf die Erreichbarkeit der Systemziele zu erwarten sind, findet eine Neuanpassung der Fahrpläne für die elektrischen Verbraucher statt. Die Optimierung der Fahrpläne in der Planungs-Phase geschieht dabei durch ein Multiagentensystem, das den *Ant-Colony* Optimierungsansatz [DB06] umsetzt.

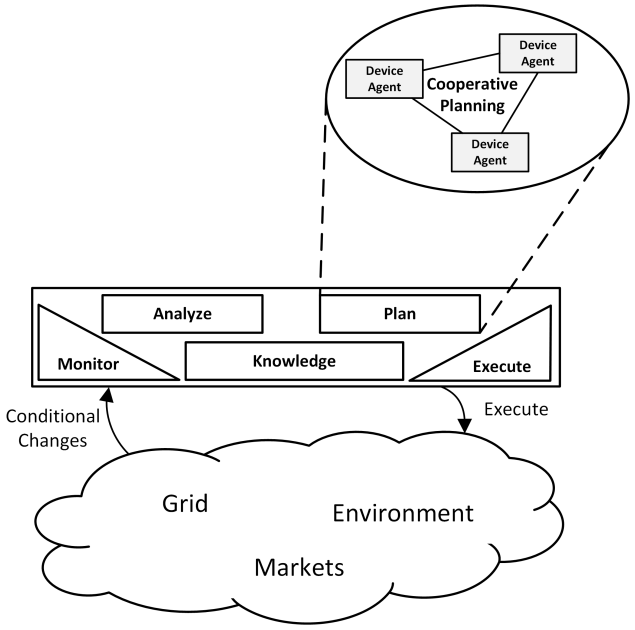



**Abbildung 2.19.:** MAPE-K Schleife zur verteilten Planung und Optimierung in einer DSM Anwendung - aus (DBP+15, S. 3).

Das System wird hierbei als eine Menge von Agenten $A$ beschrieben. In diesem Multiagentensystem repräsentiert jeder Agent $a_i \in A$ ein physikalisches Gerät, das optimiert werden soll. Jeder Agent hat dabei ein Lastprofil $L_i(n)$ und eine Menge an möglichen diskreten Laufzeiten $S_i$ innerhalb des Planungshorizonts. Eine mögliche Startzeit $s_{ic} \in S_i$ basiert dabei auf den operationalen Randbedingungen $C_i$ des physischen Geräts. Diese Randbedingungen können z.B. Benutzervorgaben oder Vorhersagen der Modelle sein. Die Auf-

gabe der Optimierung ist es, für jeden Agenten $a_i$ aus $A$ eine Laufzeit aus $S_i$ auszuwählen, sodass die Kosten der Ausführung $I$ minimiert werden. Eine generische Beschreibung der Ausführungskosten wird in [DPR14] gegeben. Im Rahmen von [DPR15a] wurde ein vereinfachtes Marktmodell verwendet, bei dem die Kosten entsprechend dem vorgegebenen Fahrplan für das zu optimierende System berechnet wurden. Ein (vereinfachter) Fahrplan ist ein Array, der ein 3-Tupel für jeden planbaren Zeitraum $k$ in der Form

$$O_k = (P_k, v_{k,low}, v_{k,high}), \tag{2.3}$$

enthält. Dabei ist $P_k$ die in der Periode $k$ angebotene Leistung zum Preis $v_{k,low}$. Wenn der geplante Verbrauch $P_{k,plannend} > P_k$ ist, kostet die Differenz $v_{k,high}$ pro Einheit. Da ein geringerer als angeforderter Verbrauch in der Regel nicht als so schädlich angesehen wird wie ein System, das den angeforderten Verbrauch übersteigt, gilt normalerweise $V_{k,low} < v_{k,high}$. Aufgrund der Elastizität der Nachfrage kann die totale Differenz zwischen $v_{k,low}$ und $v_{k,high}$ vernachlässigt werden.

Um eine vorteilhafte Struktur für die Modellierung des Problems zu erhalten, damit verschiedene kombinatorische Optimierungsalgorithmen angewendet werden können, werden die Agenten in $A$ als Liste mit einer beliebigen Permutation angeordnet. Somit kann die Anordnung der Agenten als gerichteter Graph verstanden werden. Ein Optimierer traversiert diesen Graphen und wählt aus jeder Menge $S_i$ eine mögliche Startzeit $s_{ic}$ entsprechend der Regeln des Optimierers aus. Dieses Paradigma ist ausreichend für sogenannte *semi-automatische* Geräte mit nur einer einzelnen Ausführung innerhalb des Zeitrahmens, in dem die Optimierung stattfinden soll. Zu dieser Klasse von Geräten gehören u.a. Waschmaschinen und Geschirrspülmaschinen, die nur einmal zwischen ihrer Initialisierung und der Zeit, zu der der Nutzer möchte, dass das Gerät fertig ist, laufen. Eine Randbedingung für ein solches Gerät, das von einem Agenten $a_i$ kontrolliert wird, kann z.B.

$$C_i = t_{now} < s_{ir} < (t_{end} - t_{operation}), \tag{2.4}$$

sein.

Für *vollautomatische* Geräte, die einem gewissen Zustand innerhalb bestimmter Grenzen einhalten, ist die $1:1$ Abbildung zwischen einem Agenten und der Menge aus möglichen Ausführungszeiten nicht ausreichend, um die dynamische Natur dieser Geräte abzubilden. Die erste Menge von möglichen Ausführungszeiten erlaubt normalerweise Ausführungen innerhalb der betriebsbereiten Grenzen. Abhängig von der gewählten Startzeit ergibt sich eine neue Menge von möglichen Ausführungszeiten, die den gewünschten Zustand des Geräts gewährleisten. Daher ändert sich die Anzahl der Mengen für vollautomatische Geräte dynamisch während der Optimierungsphase, in Abhängigkeit zu den gewählten Ausführungszeiten. Geräte mit mehr als einer Ausführungszeit innerhalb des optimierten Zeitrahmens haben eine $1:m$ Beziehung zwischen den Agenten und der Anzahl der Mengen von möglichen

Ausführungszeiten. Das $m$ hängt dabei von dem tatsächlichen Pfad durch den gerichteten Graphen ab.

Diese Problembeschreibung erlaubt die Optimierung der Verbraucher ohne weitere externe Ressourcen und mit minimalem Aufwand (vgl. [DPR15a]). Im Gegensatz zu Lösungsansätzen wie [CYN14] ist keine zentralisierte Laufzeitanpassung des Geschäftsmodells nötig, da die Fahrplanbestimmung im Zusammenhang mit der Preis-Elastizität der Geräte eine exakte a-priori Kontrolle des Systems erlaubt und Fehlverhalten (typischerweise Abweichungen oberhalb von $P_k$) durch höhere Preise bestraft werden. Obwohl die Komplexität des Problems exponentiell mit der Anzahl der Geräte ansteigt, steigt die Anzahl der Nachrichten innerhalb jedes Optimierungsschritts linear mit der Anzahl der Mengen $S$ an, da Broadcast- oder Multicast-Operationen vermieden werden. Eine detaillierte Beschreibung des verwendeten Ant-Colony Algorithmus findet sich in [DPR15a].

Zusammenfassend lässt sich feststellen, dass im Rahmen von [DPR15a] ein Ansatz präsentiert wurde, der zeigt, dass selbstorganisierende Eigenschaften im Allgemeinen und Selbstoptimierung im Speziellen geeignet sind, um aktuelle Herausforderung bei Modellierung von verteilten Energiemanagementsystemen zu lösen. Damit können sie einen wertvollen Beitrag bei der Lösung der Koordinations- und Kontroll-Fragestellungen liefern, die sich aufgrund des Umstiegs von traditioneller Energieerzeugung auf Basis von fossilen Brennstoffen und Kernkraftwerken hin zur Nutzung von regenerativen Energiequellen und der Formierung von *Smart Grids* im Rahmen der *Energiewende* ergeben.

## 2.6. Zusammenfassung

Die in diesem Kapitel durchgeführte Diskussion zur Konzeption selbstorganisierender Systeme hat zunächst das Potential dieses Paradigmas in Bezug auf die Entwicklung moderner, verteilter Systeme, die in der Lage sind, aktuelle Herausforderungen wie die wachsende Komplexität und unvorhersehbare Dynamiken zu bewältigen, aufgezeigt. Die genauere Analyse des Konzepts der Selbstorganisation zeigt, dass die Fähigkeit zur Selbstadaption das Hauptargument für die Attraktivität des Konzepts in vielen unterschiedlichen Anwendungsdomänen ist. Das Alleinstellungsmerkmal von selbstorganisierenden Systemen ist dabei die Art und Weise, wie diese Fähigkeit der Selbstadaptivität hergestellt wird. Dies bezieht sich konkret auf die Fähigkeit selbstorganisierender Systeme, in Abwesenheit einer zentralen Kontrollinstanz Strukturen herauszubilden oder anzupassen. Damit unterscheidet sich das Konzept der Selbstorganisation von Ansätzen wie dem *Autonomic Computing* oder *Organic Computing*, die selbstadaptives Verhalten durch hierarchische bzw. zentrale Kontrollinstanzen erzielen.

Die Attraktivität eines dezentralen Ansatzes zur Bereitstellung selbstadaptiven Verhaltens ergibt sich dabei aus den damit inhärent verbundenen nichtfunktionalen Eigenschaften dieser Systeme. Selbstorganisierende Sys-

teme zeichnen sich u.a. durch Robustheit, Skalierbarkeit, Zuverlässigkeit und eine hohe Fehlertoleranz aus. Dies sind offenkundig Eigenschaften, die für die Realisierung moderner verteilter Systeme wünschenswert sind. Diese Eigenschaften werden teilweise auf abstrakter Ebene auch als *Self-** Eigenschaften bezeichnet.

Unabhängig von der konkreten Gestaltung der Fähigkeit zur Selbstadaption zeigt sich, dass in diesem Kontext der Aspekt der Koordination von entscheidender Bedeutung ist. Aufgrund der Abwesenheit einer zentralen Kontrollinstanz in einem selbstorganisierenden System ist die Koordination ein wichtiger Baustein, um selbstadaptives Verhalten zu realisieren. Koordination ermöglicht es dabei, die lokalen Verhaltensweisen und Interaktionen der beteiligten Entitäten untereinander abzustimmen und so Veränderungen herbeizuführen. Dabei gibt es ganz unterschiedliche Formen der Interaktion, die Auswirkungen auf die Art und Weise der Koordination haben.

Insgesamt wurde in diesem Kapitel die Attraktivität des Paradigmas der Selbstorganisation aufgezeigt und anhand dreier Anwendungsbeispiele aus den Bereichen Produktionslogistik, Mobilität und Energiemanagement nochmals verdeutlicht, inwiefern Systeme aus unterschiedlichen Anwendungsdomänen auf unterschiedliche Art und Weise von selbstorganisierenden Eigenschaften, die besonders bei dem Erreichen von nichtfunktionalen Eigenschaften helfen, profitieren. Dabei zeigt sich aber auch, dass die systematische und zielgerichtete Nutzung von Selbstorganisation besondere Herausforderungen mit sich bringt. Diese liegen insbesondere an der Abwesenheit einer zentralen Kontrollinstanz und dem Bottom-up-Entwicklungsprozess. Die Modellierung, Entwicklung und der Betrieb selbstorganisierender Systeme erfordert somit neuartige Methoden und Werkzeuge, die es ermöglichen, diese besonderen Herausforderungen zu überwinden, damit die Vorteile des Paradigmas nutzbar gemacht werden können. Auf diese Herausforderungen wird nun im weiteren Verlauf dieser Arbeit genauer eingegangen.

# 3. Systematische Entwicklung selbstorganisierender Systeme

In diesem Kapitel werden die Aspekte deutlich gemacht, die für die systematische Nutzung des Paradigmas der Selbstorganisation in modernen verteilten Systemen zu beachten sind. Dabei werden zunächst die Hauptbestandteile selbstorganisierender Systeme betrachtet, um darzulegen, aus welchen Komponenten diese Systeme bestehen. Anschließend werden die spezifischen Herausforderungen bei der systematischen Entwicklung selbstorganisierender Systeme diskutiert und existierende Ansätze zur Lösung der aufgeführten Herausforderungen vorgestellt und bewertet.

## 3.1. Hauptbestandteile selbstorganisierender Systeme

Ausgehend von den Erkenntnissen aus dem vorherigen Kapitel zur Konzeption selbstorganisierender Systeme geht es in diesem Abschnitt zunächst um die systematische Entwicklung und Konstruktion selbstorganisierender Systeme. Als ersten Schritt werden dabei die Hauptbestandteile selbstorganisierender Systeme analysiert, um ein Verständnis darüber zu schaffen, welches die typischen Komponenten dieser Systeme sind. Basierend auf den bisherigen Betrachtungen zu selbstorganisierenden Systemen lässt sich ableiten, dass diese aus mindestens drei verschiedenen Arten von Komponenten bestehen. Zunächst werden Komponenten benötigt, die über die Fähigkeit, autonom zu handeln verfügen und die Kernfunktionalität des Systems umsetzen. Weiterhin werden Mechanismen benötigt, um das dynamische und adaptive Verhalten des Systems, insbesondere in Form von dezentraler Koordination, bereitzustellen und realisieren zu können. Optional ist noch eine Komponente, welche die Umgebung abbildet und steuert. Hierbei spielen insbesondere der Kontext, die Situation und die möglichen indirekten Interaktionen zwischen den Funktionskomponenten eine Rolle. Eine ähnliche Sichtweise, welche die identifizierten Hauptbestandteile um *Artefakte* ergänzt, findet sich in [MS09]:

> *SO System = Environment + Agents + SO Mechanism (rules) + Artifacts*
>
> [MS09, Seite 256]

Bei den Artefakten handelt es sich hierbei um passive Komponenten, die von der Umgebung verwaltet und von autonomen Komponenten wahrgenommen und verändert werden können [DMSFR10]. Der Begriff des Artefakts wurde ursprünglich in [RVO08] eingeführt und kann im Kontext selbstorganisierender Systeme z.B. in Form von digitalen Pheromonen zur dezentralen Koordination (basierend auf indirekter und impliziter Interaktion) von Systemkomponenten verwendet werden. Diese Arbeit folgt allerdings der Sichtweise von

[Vil13], wonach Artefakte kein eigenständiges Hauptelement bilden, sondern vielmehr als eine mögliche Realisierungsform spezieller Koordinationsmechanismen angesehen werden, die z.B. von der Umgebung verwaltet werden können.

Insgesamt kann festgestellt werden, dass in der Fachliteratur Konsens darüber besteht, dass autonome Komponenten und ihre Umgebungen integrale Bestandteile selbstorganisierender Systeme sind (vgl. beispielsweise [BCHM06, CLG+09, DMSFR10]). Dabei wird lediglich die Komponente, die wesentlich zur Selbstadaptivität dieser Systeme beiträgt, unterschiedlich bezeichnet. In [BCHM06] wird nur von einer allgemein notwendigen und vorhandenen Dynamik im System gesprochen, während [CLG+09] den Aspekt der Dynamik unter dem Begriff der weiteren Systemeigenschaften zusammenfassen. Eine solche Diversifikation bezüglich der Dynamik zur Erreichung von adaptiven Verhaltens ist nach [Vil13] folgerichtig, da genau dies der Punkt ist, in dem unterschiedliche Ansätze verfolgt werden. So unterscheiden sich in der Art und Weise, wie die Dynamik zur Selbstadaption beiträgt, auch die beiden Klassen adaptiver Systeme (selbstorganisierend und selbstadaptiv), die in Abschnitt 2.2.2 vorgestellt wurden.

### 3.1.1. Feedback-Schleifen

Eine Hauptvoraussetzung für technische Systeme, um adaptives Verhalten auszuüben, ist der Einsatz von Feedback-Schleifen [BMSG+09, CLG+09]. Mit der Hilfe von Feedback-Schleifen lässt sich das gewünschte dynamische Systemverhalten sowohl erklären als auch zielgerichtet steuern. Feedback-Schleifen sind somit eine mögliche Realisierungsform der Mechanismen, die benötigt werden, um das dynamische Verhalten des Systems herzustellen, was letztendlich zur Fähigkeit zur Adaptation führt. Ein gängiges Beispiel für eine Feedback-Schleife ist die Funktionalität eines Thermostats, welcher eine Kontrollschleife als spezielle Form einer Feedback-Schleife enthält. In den Ingenieurwissenschaften wird in diesem Kontext auch häufig von einem *Regler* gesprochen (vgl. u.a. [ZR14]). Dabei nimmt der Thermostat über Sensoren seine Umgebung war und sorgt so dafür, dass eine voreingestellte Temperatur eingehalten wird. Um dieses zu gewährleisten, werden u.a. die Öffnungen von Ventilen als eine Reaktion auf externe Ereignisse und mit dem Ziel, die vordefinierte Temperatur zu erreichen, geändert. Dieses Beispiel zeigt, dass sich mit Feedback-Schleifen Interaktionen zwischen Systemkomponenten auf Basis von kausalen Zusammenhängen beschreiben lassen.

Grundsätzlich können die Interaktionsmechanismen in einer Feedback-Schleife in zwei unterschiedliche Klassen einteilen werden. *Positives* Feedback verstärkt dabei die vorhandene Dynamik zwischen Systemelementen und kann spontan zur Bildung neuer Strukturen führen [DTc+04, Her08]. *Negatives* Feedback wiederum wirkt dem Wachstum entgegen und reguliert den dynamischen Prozess [DTc+04, Her08]. Für viele selbstorganisierende Systeme ist das Zusammenspiel zwischen positiven und negativen Feedback charakte-

ristisch [Her08]. Dabei können neue Systemstrukturen sehr schnell aufgebaut werden, bis irgendwann eine Regulierung in Form von negativem Feedback einsetzt, was dazu führt, dass das gesamte System in eine vorgesehene, stabile Struktur überführt wird. Robustheit wird dabei dadurch erreicht, dass mit diesem Prozess auch wieder neue Strukturen aufgebaut werden können, sollte dieses aufgrund von äußeren Einflüssen erforderlich sein [CDnJS+03]. Dieses Verhalten wird aus globaler Systemperspektive als Fähigkeit zur Adaption bezeichnet.

Im Bereich der naturinspirierten, selbstorganisierenden Systeme lässt sich das Zusammenspiel zwischen positiven und negativen Feedback gut am Beispiel digitaler Pheromone, die beispielsweise zur Exploration unbekannter Landschaften mit Hilfe autonomer Fahrzeuge verwendet werden können, verdeutlichen [VDPBS02]. Wenn ein Fahrzeug bei der Suche Erfolg hat und beispielsweise einen Rohstoff gefunden hat, so markiert es auf der Rückfahrt zur Ausgangsbasis den zurückgelegten Weg mit digitalen Pheromonen. Diese signalisieren dann anderen Fahrzeugen, dass hier etwas gefunden wurde, sodass die Fahrzeuge nun verstärkt in diese Richtung fahren. Damit erzeugen die Pheromone eine sich verstärkende Wirkung im System, also ein positives Feedback. Die Pheromone unterliegen aber auch den dynamischen Prozessen der Evaporation und Propagation. Dieses führt u.a. dazu, dass die Konzentration der Pheromone mit der Zeit abnimmt, bis sie letztendlich verschwindet. Dadurch wird die initiale Wirkung der Pheromone abgeschwächt. Dieses kann als negatives Feedback verstanden werden.

Damit Feedback-Schleifen systematisch und zielgerichtet verwendet werden können, ist eine formale Methodik erforderlich, die es erlaubt, die vorhandenen Dynamiken innerhalb eines Systems zu modellieren. Eine etablierte Methodik in diesem Kontext sind die *Causaul Loop Diagrams* (CLD), die auf einer grafischen Notation beruhen [Ste00]. CLD visualisieren die kausalen Zusammenhänge zwischen Systemkomponenten und identifizieren dabei positive (+) und negative (-) Feedback-Schleifen. [SR10b] verwendet die CLD-Notation, um eine Reihe weit verbreiteter naturinspirierte Koordinationsmechanismen zu analysieren und sie auf Basis ihrer dynamischen Eigenschaften zu katalogisieren. Die Modellierung der oben erwähnten Mechanismen der digitalen Pheromone zur Exploration von Ressourcen in einem unbekannten Gebiet auf Basis der CLD-Notation ist in Abbildung 3.1 dargestellt. Eine Weiterentwicklung der CLD-Notation speziell für die Modellierung von verteilten Systemen, die aus autonomen Komponenten bestehen, ist der *Agent Causal Behavior Graph* (ACBG) [RS09, SR11].

Grundsätzlich lässt sich zwischen zwei unterschiedlichen Architekturformen von Feedback-Schleifen in Systemen unterscheiden. Auf der einen Seite gibt es die zentralen und hierarchischen Formen, bei denen ein dediziertes Systemelement die Funktionalität einer Feedback-Schleife steuert. Ein Beispiel hierfür ist die weitverbreitete *MAPE-K* Schleife (siehe Abschnitt 2.2.3). Zentrale Feedback-Schleifen finden sich generell in selbstadaptiven Systemen wieder. Dem gegenüber stehen Feedback-Schleifen mit dezentraler Architektur, bei de-

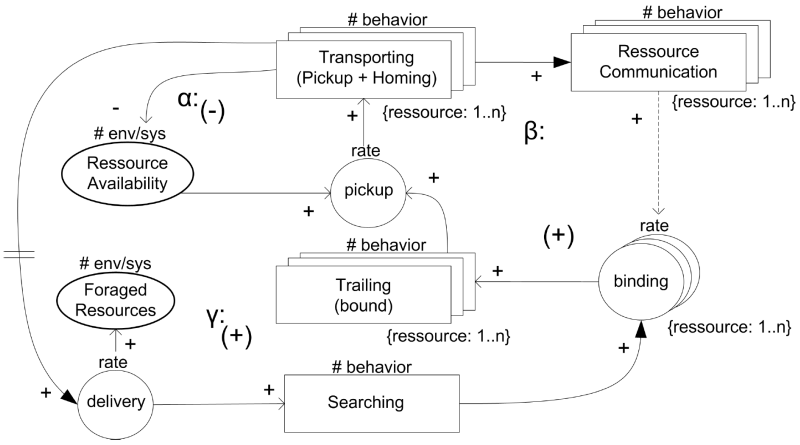



**Abbildung 3.1.:** CLD-Modell eines System mit einem Koordinationsmechanismus der auf digitalen Pheromonen basiert - aus (SR10b, S. 7).

nen die Funktionalität maßgeblich auf der Interaktion zwischen den einzelnen Systemelementen basiert. In diesem Zusammenhang spielen dezentrale Koordinationsmechanismen eine entscheidende Rolle und werden zur Verwaltung der Interdependenzen eingesetzt. Dezentrale Feedback-Schleifen sind charakteristisch für selbstorganisierende Systeme und von übergeordneter Bedeutung für deren systematische Entwicklung [BMSG+09].

### 3.1.2. Koordination

Koordination ist für selbstorganisierende Systeme von essenzieller Bedeutung. Damit sich auf makroskopischer Ebene, die vorgesehene selbstorganisierende Dynamik herausbilden kann, müssen das Verhalten und die Interaktionen der beteiligten Entitäten auf mikroskopischer Ebene entsprechend koordiniert werden. Die Auswahl und die Nutzung von Selbstorganisationsmechanismen sowie die zugrundeliegenden Koordinationsmodelle bestimmen somit maßgeblich die Funktionsweise von selbstorganisierenden Systemen. Daher muss neben der Entwicklung der eigentlichen Kernfunktionalität einer verteilten Anwendung auch deren Koordination bedacht werden. Nach [Sud10] enthält ein Koordinationsmodell drei wesentliche Bestandteile: (1) Eine Menge von Entitäten, die koordiniert werden müssen, (2) eine Menge von Medien, die zur Koordination eingesetzt werden und (3) eine formale Beschreibung des Modells selber, die semantisch beschreibt, wie die Koordination funktioniert. Die ACBG-Notation bietet zwar eine abstrakte Möglichkeit zur Erstellung von Koordinationsmodellen, diese ist allerdings für die konkrete Konstruktion, also die Programmierung, zu unspezifisch. In [SR09b] wird mit *MASDynamics* eine auf der *Extensible Markup Language*[1] (XML) basierende Beschreibungssprache vorgestellt, die es ermöglicht, Koordinationsprozesse, die z.B. als ACBG modelliert wurden, so zu beschreiben, dass sie automa-

[1] `http://www.w3.org/XML/`, aufgerufen am 1. Februar 2017

tisiert von Computersystemen verarbeitet werden können. *MASDynamics* bietet somit eine Möglichkeit, dezentrale Koordinationsstrategien zu beschreiben und die konkret an der Koordination beteiligten Entitäten sowie die für die Koordination genutzten Medien zu referenzieren.

### Programmierung

Unabhängig von konkreten Koordinationsmodellen und Systemimplementierungen stellt sich generell die Frage, welche Beziehung zwischen der Koordination und der Kernfunktionalität eines verteilten Systems besteht. Dabei war es lange Zeit üblich, bei der Implementierung solcher Systeme nicht zwischen der eigentlichen Berechnung (engl. *Computation*) und der Koordination zu trennen. Beide Aspekte wurden dabei innerhalb eines Sprachmodells verarbeitet und unterlagen somit einer engen Kopplung. Diese Praxis wurde von [GC92] in Frage gestellt und eine Trennung zwischen der eigentlichen Kernfunktionalität, der Berechnung und der Koordination in der Programmierung verteilter Systeme gefordert. Diese Idee wurde von [CGS05] speziell für die Entwicklung verteilter Systeme mit der Fähigkeit zur Selbstadaptivität aufgegriffen und eine Trennung zwischen der Kernfunktionalität und Adaptionsfunktionalität ebenfalls als wünschenswert identifiziert. Dieser grundlegende Wechsel im Progammierungsparadigma wird von *Bravetti et al.* [BGLZ05] wie folgt formuliert:

> *Programming* = *Computation* + *Coordination*
>
> [BGLZ05, Seite 1]

Dabei führt [GC92] zwei Gründe als Motivation für diesen neuen Ansatz an: Orthogonalität und Allgemeingültigkeit. Koordination soll dabei als orthogonaler Aspekt in Bezug auf die Berechnung bei der Programmierung verstanden werden. Dieses wird u.a. dadurch begründet, dass Kommunikation als wesentlicher Aspekt verteilter Systeme angesehen wird. Sprachen, welche die Berechnung beschreiben, können zwar auch Formen der Koordination enthalten, z.B. in Form von Variablen, die bei Methodenaufrufen übergeben werden, aber es ist eine Charakteristik verteilter Systeme, dass der Begriff der Koordination sich nicht nur auf den statischen Austausch von Informationen beschränkt. Vielmehr geht es bei Koordination darum, aktive und möglicherweise autonome Softwarekomponenten in einem dynamischen Umfeld mit Informationen untereinander zu versorgen. Ausgehend von dem Argument der Orthogonalität ergibt sich die Allgemeingültigkeit zwangsläufig, da, sobald der Aspekt der Koordination in ein separates Modell, z.B. eine Beschreibungssprache, ausgelagert wird, dies so gestaltet werden sollte, dass es auf möglichst viele Systeme angewendet werden kann [GC92]. Aus diesen beiden Hauptargumenten für die Trennung zwischen Koordination und Kernfunktionalität ergeben sich weitere positive Aspekte. Dabei sind Portabilität, Heterogenität, Flexibilität und Effizienz aus ökonomischer Sicht von besonderer Bedeutung [GC92]. Bewertet man die Argumente von [GC92] in Bezug auf

die Anforderungen moderner, verteilter Systeme, so wird deutlich, dass diese aktuell eine hohe Bedeutung haben, besonders im Hinblick auf die nichtfunktionalen Aspekte. Ferner ist die Argumentation von [GC92] im Allgemeinen attraktiv, um selbstorganisierende, verteilte Systeme zu implementieren, die mit aktuellen Herausforderungen wie Komplexität, Dynamik und Heterogenität konfrontiert werden.

Generell betrachtet kann der von [GC92] propagierte Paradigmenwechsel und die dort geäußerten Konzepte als Vorläufer der *Aspektorientierten Programmierung* [KH01] betrachtet werden. Aspektorientierte Programmierung befürwortet bei der Implementierung von Systemen eine Trennung von Anliegen, sodass übergreifende Aspekte eines Systems, wie z.B. Sicherheit oder auch Koordination, dediziert und getrennt von der eigentlichen Geschäftslogik behandelt werden.

#### Koordinationssprache Linda

Eine spezielle Sprache zur Beschreibung von Koordination, die den von [GC92] vorgestellten Paradigmenwechsel unterstützt, ist *Linda* [GB82]. Linda wurde im Zuge der Entwicklung des Netzwerkscomputers *Stony Brook* konzipiert. Dessen Ziel war es, verteilte Kommunikation über einen globalen Zwischenspeicher zu ermöglichen. Das Model wurde dabei als Koordinationssprache implementiert, die mehrere primitive Operationen unterstützt. Dabei wird in diesem Zusammenhang nicht von Variablen gesprochen, sondern der Begriff des *Tuple* eingeführt. Ein Tuple wird dabei über seinen Namen identifiziert und besitzt eine Liste von Parametern. Diese Tuple werden in einer abstrakten Umgebung des Systems, dem sogenannten *Tuple Space*, gehalten. Ein Linda-Programm ist letztendlich eine Sammlung geordneter Tuple, auf denen die folgenden Operationen ausgeführt werden können:

- **in**: Eine atomare Operation, die ein Tuple liest und es vom Tuple Space entfernt (konsumiert).
- **rd**: Ein nicht destruktives Lesen eines Tuples.
- **out**: Produziert ein Tuple und schreibt es in den Tuple Space.
- **eval**: Erzeugt einen neuen Prozess um Tuple zu evaluieren und schreibt die Ergebnisse in den Tuple Space.

Linda besteht im Kern aus eher einfachen Bestandteilen, die aber in ihrer Summe einen vielseitigen Ansatz ergeben, der sich insbesondere durch Flexibilität und effiziente Implementierung hervorhebt [GB82]. Da die *in* und *out* Operationen voneinander entkoppelt durchgeführt werden, wird eine asynchrone Kommunikation unterstützt. Darüber hinaus unterstützt das Konzept der Tuple Spaces eine asynchrone und räumliche Entkopplung [GB82]. Dieses sind zwei Eigenschaften, die auch für moderne verteilte Systeme von großer Bedeutung sind. Daher ist es auch nicht weiter verwunderlich, dass Linda auch

heute noch für verschiedene Systeme und Anwendungsgebiete weiterentwickelt und verwendet wird:

- *SwarmLinda* [CMT04] ist eine Spezialisierung für Systeme mit Schwarmverhalten.
- *ObjectPlaces* [SH04] setzt Tuple Spaces in Multi-Agenten Systemen mit explizierter Modellierung ihrer Umgebung ein.
- *Tuples On the Air* [MZ06b] wurde im Hinblick auf die speziellen Anforderungen in mobilen und ubiquitären Umgebungen entwickelt.
- Weiterhin gibt es zahlreiche Bibliotheken, um Linda in unterschiedlichen Programmiersprachen, wie z.B. C[2], C++[3], Phyton[4] oder Ruby[5], einzusetzen.

### 3.1.3. Agenten und Aktive Komponenten

Agenten bzw. Multi-Agenten Systeme (MAS) sind ein weit verbreiteter Ansatz zur Realisierung komplexer verteilter Systeme (vgl. u.a. [VSL+10]). Multi-Agenten Systeme sind dabei ein relativ junges Teilgebiet der Informatik, dessen Wurzeln u.a. in der verteilten künstlichen Intelligenz liegen. Das Agentenparadigma ist für moderne verteilte Systeme im Allgemeinen und für selbstorganisierende Systeme im Speziellen von großer Bedeutung. Insgesamt stellt das Agentenparadigma ein Konzept dar, das die Entwicklung adaptiver Systemkomponenten ermöglicht, die in der Lage sind, über ihren Zustand zu *deliberieren* und in Abhängigkeit davon künftige Aktionen auszuwählen bzw. auszuführen. *Deliberative* Agentenarchitekturen zeichnen sich dadurch aus, dass vor der tatsächlichen Durchführung bzw. Auswahl einer Aktion zunächst die möglichen Alternativen in Betracht gezogen und bewertet werden [SL01]. Somit ist es Entwicklern möglich, ein System auf konzeptioneller Ebene mit verschiedenen Strategien auszustatten und zur Laufzeit die Effekte des Verhaltens zu studieren. Aufgrund ihrer inhärent verteilten und dezentralen Architektur eignen sich Multi-Agenten Systeme besonders gut für die Entwicklung von verteilten, selbstorganisierenden Systemen und sind in der Lage, die nichtfunktionalen Anforderungen solcher Systeme gut zu erfüllen [PVR13].

Aktuell gibt es zahlreiche, unterschiedliche Definitionen darüber, was einen Agenten ausmacht (z.B. [Bra07, Pok07]), allerdings basieren diese in den meisten Fällen auf Agenteneigenschaften, wie sie von [WJ95] benannt wurden:

- **Autonomie** bezeichnet die Eigenschaft eines Agenten, selbstständig zu handeln. Agenten benötigen keine Intervention von externen Entitäten und besitzen stattdessen Kontrolle über ihre Aktionen und ihren internen Zustand.

[2] `http://linuxtuples.sourceforge.net/`, aufgerufen am 1. Februar 2017
[3] `https://sourceforge.net/projects/cpplinda/`, aufgerufen am 1. Februar 2017
[4] `https://code.google.com/archive/p/pylinda/`, aufgerufen am 1. Februar 2017
[5] `http://ruby-doc.org/stdlib-2.3.0/libdoc/rinda/rdoc/index.html`, aufgerufen am 1. Februar 2017

- **Soziale Fähigkeiten** bezeichnet die Fähigkeit eines Agenten, mit anderen Agenten zu interagieren, i.d.R. werden dafür spezielle Kommunikationssprachen verwendet.
- **Reaktivität** bezeichnet die Eigenschaft, dass Agenten ihre Umgebung wahrnehmen können und auf Veränderungen in dieser angemessen reagieren können.
- **Proaktivität** bezeichnet die Eigenschaft, dass Agenten nicht nur auf Änderungen in ihrer Umgebung reagieren können, sondern vielmehr auf eigene Initiative aktiv und zielgerichtet agieren können.

Auch wenn diese Aufzählung keinen Anspruch auf Vollständigkeit erhebt, so charakterisiert sie doch die wesentlichen Aspekte von Agenten. Ergänzend dazu wurde in [FG97] eine Taxonomie von Agenten aufgestellt, die die unterschiedlichen Domänen unterstreicht, in denen Software-Agenten eingesetzt werden (siehe Abbildung 3.2). Ein weiteres wichtiges Anwendungsfeld von Multiagentensystemen, das auch im Rahmen dieser Arbeit eine wichtige Rolle spielt, sind Simulationen (vgl. [Bra07, Pok07, YÖ09, UW09]).

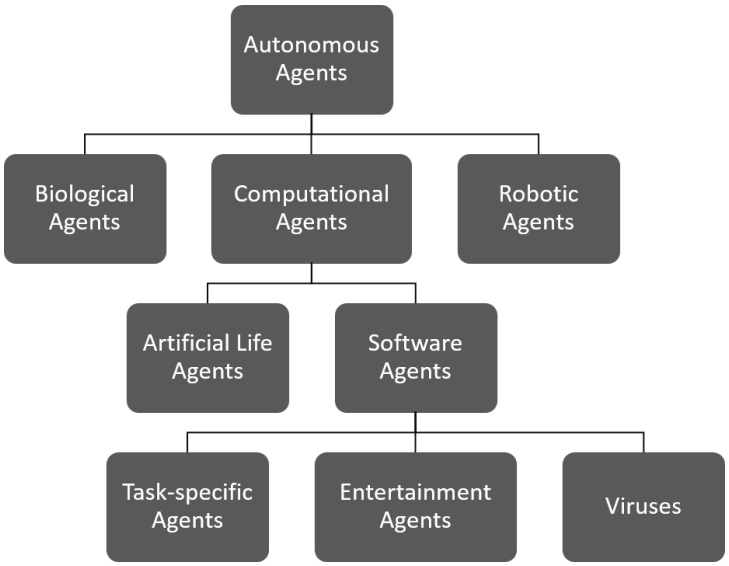



**Abbildung 3.2.:** Taxonomie autonomer Agenten - nach (FG97).

Neben ihren Einsatzzweck lassen sich Agenten auch auf Basis ihrer internen Architekturen unterscheiden, wobei es natürlich einen Zusammenhang zwischen dem Einsatzzweck eines Agenten und seines internen Aufbaus gibt. Im Rahmen dieser Arbeit sind dabei zwei Formen von besonderen Interesse: Das *Belief-Desire-Intention* (BDI) Modell und das *Task-Modell*. Das BDI-Modell von *Bratman* [Bra87] beruht dabei auf einer philosophischen Theorie und versucht, das rationale Handeln von Menschen auf die Modellierung von Agenten abzubilden [Bra07]. Dabei geht das BDI-Modell davon aus, dass Menschen ihr Wissen über die Welt in ihren *Beliefs* abspeichern und diese aktiv in ihr Handeln einbeziehen. Ein Agent, der diesem Modell folgt, handelt somit auf Basis seiner

Wissens-/Glaubensbasis. Darüber hinaus verfügt ein solcher Agent über eine Reihe von *Desires*, die globale Ziele darstellen, deren Erreichung für den Agenten als wünschenswert gilt. Schließlich gibt es noch die *Intentions*, die konkrete Pläne abbilden, welche ausgeführt werden, um die globalen Ziele zu erreichen. Das BDI-Modell stellt somit einen Ansatz dar, mit dem sich komplexes und autonomes Verhalten in Agenten modellieren und implementieren lässt. Es ist somit besonders für Anwendungen geeignet, für die die Fähigkeit des proaktiven Handelns wichtig ist. Im Gegensatz dazu eignet sich das Task-Modell eher für Anwendungsfälle, bei denen die Komplexität gering ist und das reaktive Handeln der Agenten im Vordergrund steht. Das Task-Modell geht dabei von dem Verhalten eines Agenten aus, das sich aus einer oder mehrerer Aufgaben zusammensetzt, die teilweise auch parallel ausgeführt werden können [Bra07]. Natürlich gibt es über diese beiden Ansätze hinaus noch weitere Theorien und Architekturen, die den Aufbau und das Verhalten von Agenten beschreiben; dieses schließt u.a. auch Mischformen mit ein, bei denen die beiden vorgestellten Ansätze kombiniert werden. Eine umfangreichere Analyse über die verschiedenen Formen der Agenten-Architekturen findet sich u.a. in [PBL05].

### Aktive Komponenten

Eine jüngere Entwicklung, die das Agentenparadigma erweitert, sind die sogenannten *Aktiven Komponenten* [PBJ10, BP12b, PB13]. Obwohl das Agentenparadigmen gut geeignet ist, um verteilte Systeme mit selbstorganisierenden Fähigkeiten zu entwickeln, so vernachlässigt eine Beschränkung auf dieses Paradigma doch andere etablierte Ansätze wie Service-orientierte Architekturen (SOA) [PvdH07], Prozesse und Workflows [vHadA02] sowie Service-Komponenten-Architekturen [MR09]. Eine Übersicht über verschiedene Paradigmen zur Entwicklung von komplexen verteilten System mit ihren jeweiligen Stärken und Schwächen sowie den typischen Anwendungsgebieten wird in Abbildung 3.3 dargestellt.

Das Paradigma der Aktiven Komponenten basiert auf der Erkenntnis, dass eine Kombination bestehender Paradigmen und ihrer jeweiligen Vorteile bei der Entwicklung komplexer verteilter Systeme nützlich sein kann. Abbildung 3.4 zeigt dabei, wie das Konzept der Aktiven Komponenten zunächst das Agentenparadigmen mit der Service Komponenten-Architektur vereint. Daraus ergibt sich, dass eine Aktive Komponente über Service-Schnittstellen von anderen Entitäten aufgerufen werden und so Dienste anbieten kann. Die Service-Schnittstellen verbergen dabei die interne Architektur der Aktiven Komponenten, sodass es von außen nicht ersichtlich ist, auf Basis welchen Paradigmas die Geschäftslogik realisiert ist. Intern kann eine Aktive Komponente also z.B. einen Workflow enthalten oder auch ein BDI-Agent sein. Durch die Bildung von Sub-Komponenten können Aktive Komponenten verschachtelt werden, um so komplexere Strukturen zu bilden. Das Paradigma der Aktiven Komponenten ist auf Grund seiner Flexibilität und der Einbindung aktueller Entwicklungsstrategien für lose gekoppelte, komplexe verteilte Systeme auch für diese

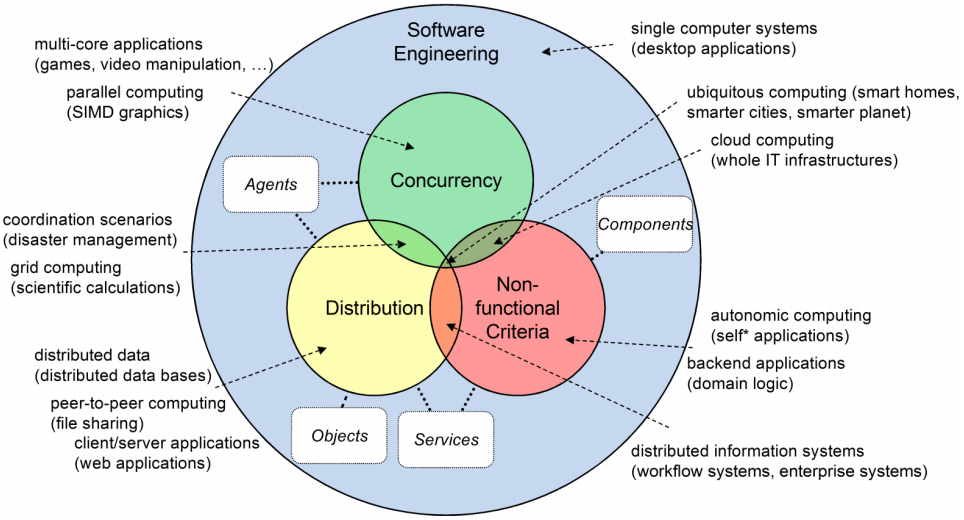



**Abbildung 3.3.:** Paradigmen zur Entwicklung komplexer verteilter Systeme mit ihren jeweiligen Stärken und den typischen Anwendungsgebieten - aus (BP12b, Seite 3).

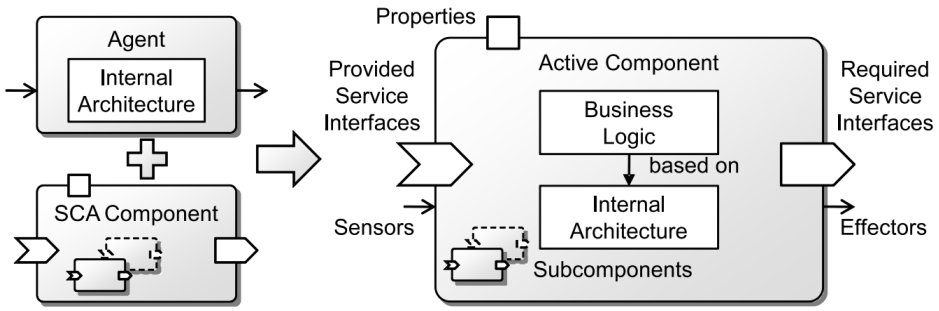



**Abbildung 3.4.:** Konzeptionelles Modell einer Aktiven Komponente - aus (BP12b, Seite 5).

Arbeit von großem Interesse. Es vereint dabei die identifizierten, nützlichen Eigenschaften von Agenten und erweitert sie um ebenfalls nützliche Entwicklungen aus dem Gebiet der serviceorientierten Architekturen. Im Zusammenhang mit dem Paradigma der Aktiven Komponenten ist das Open-Source Framework *Jadex* [BP12a] von besonderer Bedeutung. *Jadex* ermöglicht die Programmierung verteilter Anwendungen auf Basis von Aktiven Komponenten und unterstützt die Entwickler dabei mit reichhaltiger Werkzeugunterstützung. Durch seine Unterstützung von Aktiven Komponenten und damit sowohl BDI-Agenten als auch reaktiven, task-orientieren Agenten wird *Jadex* im Rahmen dieser Arbeit an mehreren Stellen zur Realisierung von selbstorganisierenden System verwendet.

### 3.1.4. Umgebung

Das Konzept der *Umgebung* stammt neben dem Kernaspekt der Agenten aus dem Umfeld der Multiagentensysteme. Die Autoren von [WJ95] sprechen dabei in ihrer Definition von Multiagentensysteme von einer Umgebung, in der die

Agenten situiert sind. Die Definition von [OPF03] geht an der Stelle noch einen Schritt weiter und behauptet, dass ein Agent ohne modellierte Umgebung im Grunde nutzlos ist. Diese Verständnis wird in den Arbeiten von [SH04, WSR+05, KG06] aufgriffen und weiter entwickelt, mit dem Ergebnis, dass vorgeschlagen wird, bei der Entwicklung von Multiagentensystemen Umgebungen als Design- und Implementationsobjekte erster Ordnung zu verstehen. Dabei wird u.a. die Forderung abgeleitet, explizite Konzepte zu entwickeln, die sich auf Aspekte abseits der Kernfunktionalitäten von Agenten fokussieren. Die Umgebung eines Multiagentensystems bietet eine Möglichkeit, um das in Abschnitt 3.1.2 vorgestellte Konzept der Trennung zwischen Koordinations- und Kernfunktionalität adäquat auf moderne verteilte Systeme zu übertragen, die auf dem Paradigma der Multiagentensysteme beruhen.

Eine Beschreibung der charakteristischen Eigenschaften einer Umgebung findet sich in der Definition von [RN04]. Hier werden einige Dimensionen aufgelistet nach denen sich die Kennzeichnen einer Umgebung allgemein, also unabhängig von einer bestimmten Domäne, beschreiben lassen:

- **Vollständig oder teilweise beobachtbar:** Es wird von vollständiger Beobachtbarkeit gesprochen, wenn ein Agent zu jedem Zeitpunkt über seine Sensoren auf den vollständigen Zustand der Umgebung zugreifen kann. Wenn Umgebungen mit dieser Eigenschaft vorliegen, müssen die Agenten keine interne Repräsentation über den Zustand der Umgebung abspeichern. Von teilweise beobachtbaren Umgebungen wird gesprochen, wenn die Sensoren eines Agenten ungenau, unvollständig oder fehlerhaft arbeiten.

- **Einzel-Agenten- oder Multi-Agenten Umgebung:** Der grundlegende Unterschied zwischen Einzel-Agenten- und Multi-Agenten-Umgebungen lässt sich selbsterklärend aus deren Namen ableiten. Für Multi-Agenten-Umgebungen ergibt sich aber ein weiteres interessantes Unterscheidungsmerkmal: Hierbei kann zwischen *kooperativen* und *konkurrierenden* Umgebungen unterschieden werden. Bei Schachspiel ist beispielsweise der letztere Fall gegeben, demgegenüber kann das Zusammenwirken einer Fußballmannschaft als Manifestation einer kooperativen Multi-Agenten-Umgebung aufgefasst werden.

- **Deterministisch oder stochastisch:** Von einer deterministischen Umgebung spricht man, wenn sich der nächste Zustand der Umgebung komplett und eindeutig aus dem aktuellen Zustand und den von den Agenten durchgeführten Aktionen ergibt. Ist dies nicht der Fall, wird von einer stochastischen Umgebung gesprochen. Die meisten Situationen, die sich in der Realität ergeben, sind so komplex und dynamisch, dass diese Umgebungen als stochastisch zu betrachten sind.

- **Episodisch oder sequenziell:** Eine Episode bezeichnet in diesem Zusammenhang eine atomare Aktion eines Agenten. Diese besteht aus einer Wahrnehmung und einer ausgeführten Aktion. Wenn die aktuelle

Episode nicht mit der vorangegangen zusammenhängt, wird von einer episodischen Umgebung gesprochen; ist das Gegenteil der Fall, von einer sequenziellen Umgebung.

- **Statisch oder dynamisch:** Wenn sich der Zustand einer Umgebung ändern kann, während der Agent über seine nächste Aktion entscheidet, wird von von einer dynamischen Umgebung gesprochen, andernfalls von einer statischen.
- **Diskret oder kontinuierlich:** Die Unterscheidung zwischen einer diskreten und einer kontinuierlichen Umgebung bezieht sich auf den Zustand der Umgebung, sein Zeitmodell und die Wahrnehmung und Aktionen von Agenten. Wenn die Menge der unterschiedlichen Zustände dabei endlich ist, wird von einer diskreten Umgebung gesprochen, im anderen Fall von einer kontinuierlichen.

Die Aufzählung dieser Dimensionen verdeutlicht die unterschiedlichen Möglichkeiten, die sich bei der Modellierung einer Umgebung ergeben. Ausgehend auf diesen aufgestellten Dimensionen ist es nun auch möglich, die Rolle einer Umgebung in einem Multiagentensystem genauer zu spezifizieren. Die Autoren von [WSR+05] identifizieren hierbei sechs grundlegende Aufgaben, die auf der bereits beschriebenen Sichtweise beruhen, Umgebungen als eigenständige Design- und Implementationsobjekte erster Ordnung zu verstehen:

1. **Strukturierung:** Da eine Umgebung ein gemeinsam genutzter Raum für die Agenten, Ressourcen und Services ist, strukturiert sie das ganze System. Dabei gibt es unterschiedliche Möglichkeiten, wie diese Strukturierung aussehen kann. In diesem Zusammenhang spielen räumliche [BMV05] und organisatorische [ZJW03] Aspekte eine Rolle. Außerdem kann die Umgebung auch die Rolle eines Koordinationsmediums einnehmen [JR02].
2. **Ressourcen- und Service-Management:** Die Umgebung ermöglicht und kontrolliert gleichermaßen den Zugriff auf Ressourcen und Services. Services bieten in diesem Zusammenhang auf Nachfrage Funktionalitäten für die Agenten an, während Ressourcen von den Agenten wahrgenommen, verändert und konsumiert werden können.
3. **Beobachtbarkeit:** Eine Umgebung muss zwingend beobachtbar sein, d.h. die Agenten müssen in der Lage sein, ihre Nachbarschaft zu observieren. Ferner sollte es für einen Agenten möglich sein, die Aktionen anderer Agenten in der Umgebung wahrnehmen zu können. Die Autoren von [WSR+05] zählen dabei zum Aspekt der Beobachtbarkeit der Umgebung auch die semantische Beschreibung der Anwendungsdomäne, etwa durch die Definition einer Ontologie (vgl. [CCC+05]).
4. **Kommunikation:** Ein intrinsischer Bestandteil von Multiagentensystemen ist die Kommunikation. Die Umgebung kann hierbei vorgeben,

welche unterschiedlichen Kommunikationsmöglichkeiten genutzt werden können. Hierbei bieten sich neben der häufig verwendeten nachrichtenbasierten Form auch indirekte Formen an (vgl. Abschnitt 2.3.2). Dabei platzieren Agenten Nachrichtenobjekte in der Umgebung, die wiederum von anderen Agenten konsumiert werden können.

5. **Prozesse:** Die Umgebung selbst kann ebenfalls Aktivitäten bezogen auf Ressourcen ausführen und damit eine aktive Rolle im System einnehmen. So kann die Umgebung z.B. die Evaporation und Propagation digitaler Pheromone steuern [VDPBS02] und aktiv für eine Dynamik sorgen. Solche Dynamiken können als Prozesse der Umgebung verstanden werden, da sie den Gesamtzustand des Systems verändern und insbesondere für selbstorganisierende Systeme von Nutzen sind.

6. **Regeln und Normen:** Eine weitere Aufgabe der Umgebung kann die Erstellung von Regeln und Normen sein, die das Verhalten der Agenten und Ressourcen festlegen. Dadurch können z.B. Konsistenzbedingungen im System durchgesetzt oder auch die Nutzung von Ressourcen durch die Agenten geregelt werden.

Auch wenn in der vorangegangenen Aufzählung der Aspekt der Koordination nicht explizit aufgeführt ist, so lässt sich dieser doch aus den aufgeführten Aspekten bzw. einer Kombination dieser ableiten. So beschäftigen sich z.B. die Autoren von [SH04] mit Umgebungen, deren Zweck ausdrücklich der Koordination dient. Dabei werden u.a. die funktionalen und nicht-funktionalen Eigenschaften solcher Koordinationsumgebungen charakterisiert und mehrere konkrete Beispiele genannt, in denen die Umgebung zur Koordination eingesetzt wird. Ein weiteres Beispiel für die Verwendung der Umgebung zur Koordination findet sich in [MZ06b]. Hier werden die Bewegungen von Agenten in einem mobilen Netzwerk mit Hilfe der Umgebung koordiniert. In der Arbeit von [Brü00] wird die Umgebung genutzt, um auf Basis von Ameisenalgorithmen Fabrikanlagen zu steuern. Der im Kontext der Koordination wichtige Aspekt der Kommunikation unter Berücksichtigung der Umgebung wird in der Arbeit von [OPF03] ausführlich diskutiert.

Werden die bisherigen Ausführungen zum Aspekt der Umgebung betrachtet, so zeigen sich die vielfältigen Einsatzmöglichkeiten und Dimensionen dieses Konzeptes. Daher sprechen sich die Autoren von [GM05] dafür aus, mehrere Umgebungen in einem System zu nutzen. Dabei wird die bis dahin gängige Praxis, verschiedene Aspekte in einer einzigen Umgebung gemeinsam zu verwalten, in Frage gestellt. Daher wird in [GM05] eine konzeptionelle Erweiterung vorgeschlagen, die das Ziel verfolgt, mit mehreren Umgebungen sowohl Modularität als auch die Trennung von logischen Zuständigkeiten zu erreichen, um damit insbesondere die Entwicklung von komplexen, verteilten Systemen zu unterstützen. Dieses wird erreicht durch eine Konzentration auf die Charakteristika der Beziehungen zwischen den Agenten und der Umgebung. Diese Eigenschaften beinhalten die Ontologie der Umgebung

und die sich daraus ergebenen Möglichkeiten der Perzeption und Aktion für die Agenten sowie die resultierenden Möglichkeiten der Interaktion. Diese Generalisierung der Beziehung zwischen Agent und Umgebung ermöglicht eine zielgerichtete Instanziierung und damit auch die Nutzung mehrerer Umgebungen, da ein Agent in mehreren Umgebungen existieren kann und in jeder Umgebung klar definiert ist, welche Möglichkeiten zur Interaktion gegeben sind. Diese logische Trennung von Aspekten durch jeweils separate Umgebungen und die damit einhergehende Option zur nebenläufigen Verwaltung ist im Bezug auf die Entwicklung selbstorganisierende Systeme ein wünschenswerter Aspekt und wird daher auch im weiteren Verlauf dieser Arbeit eine Rolle spielen.

## 3.2. Entwicklung selbstorganisierender Systeme

Wie bereits im Rahmen dieser Arbeit festgestellt wurde, beinhaltet das Paradigma der Selbstorganisation viele Eigenschaften, die wünschenswert für moderne verteilte Systeme sind, um diese in die Lage zu versetzen, mit aktuellen Herausforderungen entsprechend umzugehen. Diese Charakteristika beziehen sich dabei insbesondere auf nichtfunktionale Eigenschaften wie z.B. Robustheit und Skalierbarkeit. Allerdings bringt das Paradigma der Selbstorganisation auch spezielle Eigenschaften mit sich, die die systematische Entwicklung entsprechender Systeme vor neuartige Herausforderungen stellt. In diesem Abschnitt geht es darum, zunächst die charakteristischen Herausforderungen der Entwicklung von selbstorganisierenden Systemen vorzustellen, um danach existierende Ansätze zu diskutieren, die versuchen diese Herausforderungen zu lösen.

### 3.2.1. Herausforderungen

Bereits im vorangehenden Kapitel wurden bei der Vorstellung der Konzepte selbstorganisierender System erste spezifische Herausforderungen genannt. Dazu gehören insbesondere nichtlineares und nichtdeterministisches Verhalten. Im Hinblick auf diese Eigenschaften selbstorganisierender Systeme gibt es grundsätzlich zwei verschiedene Standpunkte. Einerseits ermöglichen es gerade diese Eigenschaften, Systemeigenschaften zu erzielen, die Selbstorganisation für moderne verteilte Systeme interessant machen. Andererseits ergibt sich dadurch das Problem, dass etablierte Methoden und Ansätze zur Systementwicklung Schwierigkeiten haben, solche Systeme mit den genannten Eigenschaften systematisch zu entwickeln. Das bereits bekannte Beispiel der Futtersuche von Ameisen mit Hilfe von Pheromonen eignet sich gut, um die Vor- und Nachteile dieser Eigenschaften zu demonstrieren (vgl. [Vil13]). Eine Ameise, die ausgehend vom Nest mit der Futtersuche beginnt, wählt mit einer gewissen Wahrscheinlichkeit den nächsten Weg aus. Bei der Auswahl orientiert sie sich dabei an der Stärke der wahrgenommenen Pheromone, wählt aber nicht automatisch den Weg mit der stärksten Konzentration. Durch dieses Verhal-

ten werden immer wieder neue Wege zu neuen Futterquellen gefunden, sodass blockierte Wege, z.B. durch einen umgefallenen Baumstamm, durch diese ersetzt werden können. Auf globaler Ebene zeigt das System so ein adaptives und robustes Verhalten, eine Eigenschaft, die generell als wünschenswert angesehen wird. Gleichzeitig stellt sich die Frage, wie dabei sichergestellt werden kann, dass alle Futterquellen in der Umgebung in einer bestimmten Zeit gefunden und abgebaut werden können. Werden nun die bisher genannten spezifischen Schwierigkeiten bei der systematischen Entwicklung verteilter, selbstorganisierender Systeme mit einbezogen, so lassen sich generell folgende Herausforderungen identifizieren:

- Bottom-Up-Entwurf (vgl. Abschnitt 2.2.2).
- Nichtlineare Eigenschaften und nichtdeterministisches Verhalten (vgl. Kapitel 2)
- Herausbildung emergenter Strukturen und ggf. ungewollter emergenter Phänomene (vgl. Abschnitt 2.1.2)
- Fehlende formale Methoden zur Überprüfung und Analyse des Laufzeitverhaltens (vgl. [KGGP09])

Ein wesentlicher Punkt für viele der aufgeführten Herausforderungen ist dabei die Abwesenheit einer zentralen Kontrollinstanz, die in Kombination mit der Lokalität der Interaktion von Systemelementen in selbstorganisierenden Systemen einen Bottom-Up-Entwicklungsprozess bedingt. Dieser wiederum steht im Gegensatz zu klassischen Top-down-Ansätzen, bei denen zunächst die globalen Systemziele definiert werden, die dann auf unteren Ebenen verfeinert und umgesetzt werden. Eine weitere Schwierigkeit ergibt sich aus der Wahl geeigneter Systemparameter auf mikroskopischer Ebene, sodass auf makroskopischer Ebene das vorgesehene Systemverhalten erzielt werden kann. Daraus ergibt sich ein erheblicher Aufwand für die systematische Verbindung zwischen den Wirkungsweisen dieser beiden Systemebenen [MBAG06]. Diese beiden Systemebenen werden in diesem Zusammenhang von [HWMH06] auch als *Komponenten-* und *Gruppenebene* bezeichnet. Dabei betont erstere die Sichtweise auf einzelne Systementitäten, während zweitere die kollektive Betrachtung des sich ergebenden Verhaltens der interagierenden Systementitäten im Vordergrund hat. Ein Ansatz zur systematischen Verbindung der beiden Systemebenen wird in [SR08] durch Einführung einer *mesoskopischen* Modellierungsebene vorgeschlagen. Diese soll dabei als Mediator zwischen den beiden Ebenen fungieren und versucht die Transitionen zwischen makroskopischen Zuständen durch Artefakte zu erklären, die auf der Mediatorenebene angesiedelt sind. Die Artefakte abstrahieren dabei von den einzelnen mikroskopischen Zuständen, indem sie das mittlere, durchschnittliche Verhalten beschreiben.

Ein weiterer wichtiger Punkt im Zusammenhang mit der systematischen Verbindung zwischen mikroskopischer und makroskopischer Ebene sind emergente Phänomene. Dazu liefert [Mog06] eine Aufzählung, die verdeutlicht, dass

eine systematische Entwicklung selbstorganisierende Systeme wichtig ist, um emergente Phänomene zu vermeiden, die ein nicht-vorgesehenes Systemverhalten erzeugen würden:

- **Flattern**: Dieser Effekt, der in der englischen Fachliteratur als *Thrashing* gezeichnet wird, kann eintreten, wenn mehrere Komponenten in konkurrierender Art und Weise auf eine gemeinsame Ressource zugreifen und dabei eine unverhältnismäßig hohe Zeit darauf verwendet wird, den Zugriff auf die Ressource zu regeln, sodass die Komponenten von der Verrichtung ihrer eigentlichen Aufgaben abgehalten werden. In diesem Fall wird der durch das ständige Umschalten auftretende Effekt als *Flattern* bezeichnet.

- **Unerwünschte Synchronisation**: Hiervon ist die Rede, wenn eine Menge von Systemen, deren zeitliches Verhalten ursprünglich in keiner Korrelation steht, nach einer Weile ein korrelierendes Verhalten aufweisen, obwohl dies nicht vorgesehen ist.

- **Unerwünschte Oszillation**: Dieser Effekt bezeichnet den Zustand, wenn ein System aufgrund einer zufälligen oder beim Design nicht korrekt modellierten Feedback-Schleife zwischen mehreren Systemzuständen oszilliert.

- **Deadlock**: Hiervon ist die Rede, wenn Systemkomponenten bei dem Zugriff auf eine gemeinsame Ressourcen so in Konflikt miteinander geraten, dass in Bezug auf die Abhängigkeiten ein Zyklus entsteht, der die weitere Verarbeitung im System blockiert.

- **Livelock**: Bei diesem Effekt findet kein Fortschritt im System statt, da mindestens zwei Komponenten sich gegenseitig behindern. Man unterscheidet den *Livelock* von *Deadlock* dadurch, dass im Falle des *Livelocks* das System an sich nicht blockiert ist, da die einzelnen Komponenten noch erreichbar sind.

- **Phasenwechsel**: Hiervon spricht man, wenn sich das Verhalten des Systems als Reaktion auf eine kleine Parameteränderung radikal ändert. In diesem Fall spricht man auch von einer erhöhten *Sensitivität* des Systems.

- **Chaotisches Verhalten**: Hiervon ist die Rede, wenn als Ergebnis einer schwer vorhersehbaren Dynamik, einer plötzlichen Änderung im Verhalten und einer überproportional starken Veränderung einer ursprünglich schwachen Dynamik das resultierende Systemverhalten chaotisch wirkt.

Die vorgestellten Schwierigkeiten und Herausforderungen treten u.a. dadurch auf, dass selbstorganisierende Systeme formal nur schwer zu analysieren sind. Dadurch resultiert als Konsequenz, dass andere Methoden gefunden werden müssen, mit denen im Laufe des Entwicklungsprozesses

sichergestellt werden kann, dass das System sich so verhält, wie es vorgesehen ist. Systemsimulationen und systematische Evaluierungsmethoden spielen dabei eine wichtige Rolle. Im Kontext der Analyse des Systemverhaltens gilt es dabei zu bedenken, dass die Performanz der Kernfunktionalität nur einen Teilaspekt bildet. Darüber hinaus besteht die Notwendigkeit, auch die Qualität der Erfüllung nichtfunktionaler Anforderungen zu quantifizieren. Ein genereller Faktor, der unabhängig von den bisher konkret aufgeführten, spezifischen Schwierigkeiten bei der systematischen Entwicklung verteilter selbstorganisierender Systemen auftritt, ist die *Unsicherheit* [RJC12]. Diese betrifft alle Phasen der Systementwicklung. In [RJC12] wird die Thematik der Unsicherheit ausgiebig analysiert, indem u.a. eine Taxonomie vorgeschlagen und zudem für einzelne Phasen der Systementwicklung konkret spezifiziert wird, welche Unsicherheitsfaktoren potenziell möglich sind. Dies stützt die These, dass die zielgerichtete Entwicklung verteilter selbstorganisierender Systeme eine besondere Herausforderung darstellt [HWMH06, CLG+09, VS09].

### 3.2.2. Prinzipielle Entwicklungsstrategien

Bei der Entwicklung selbstorganisierender Systeme spielt die Koordinationslogik eine entscheidende Rolle. Dabei lassen sich unterschiedliche Strategien finden, die sich danach unterscheiden lassen, wie die Koordinationslogik mit der Kernfunktionalität zusammenhängt. In der Dissertation von *Sudeikat* [Sud10] werden hier drei grundlegende Ansätze unterschieden:

1. Nicht-ausgelagerte Koordination
2. Ausgelagerte Koordination
3. Partiell ausgelagerte Koordination

Diese drei Ansätze werden im Folgenden genauer diskutiert. Ein besonderer Fokus liegt auf den Auswirkungen, die sich dabei auf den Entwicklungsprozess ergeben.

#### Nicht-ausgelagerte Koordination

Bei konventionellen Entwicklungsstrategien wird nicht zwischen der Kernfunktionalität der Komponenten und ihrer Koordination unterschieden. Dementsprechend wird diese Strategie als nicht ausgelagerte Koordination bezeichnet. Dieses Vorgehen bietet sich dann an, wenn schon in einem frühen Stadium des Entwicklungsprozesses bekannt ist, über welche dynamischen Eigenschaften die Anwendung verfügen wird. In diesem Fall kann der Entwickler direkt den Aspekt der Koordination in das Design der Komponenten einbeziehen. Hierfür muss der konventionelle Entwicklungsprozess um Aktivitäten erweitert werden, die sich mit der Konzeption und der Implementierung der Koordinationsfunktionalität beschäftigen. Die wesentliche Charakteristik der nichtausgelagerten Koordination liegt darin, dass die Kernelemente der Koordination direkt in die Modelle der Systemkomponenten eingefügt werden. Die Ker-

nelemente sind diejenigen Elemente, welche die Interaktion zwischen Komponenten realisieren und zudem für die lokalen Anpassungen der Komponenten selber verantwortlich sind. Während der Entwicklung kann z.B. durch die Simulation von Modellen des Systems überprüft werden, ob die vorgesehene Dynamik erzieht wird, sodass die Konfiguration der Komponenten ggf. angepasst werden kann (siehe z.B. [RPS12]). Die Strategie der nicht-ausgelagerten Koordination hat zur Folge, dass die Koordinations- und Kernfunktionalität eng miteinander verflochten sind. Dadurch ist der Wechsel von konkreten Interaktionsmechanismen oder auch Koordinationsmuster sehr aufwendig und zeitintensiv. Daher ist diese Strategie ungeeignet für Anwendungsdomänen, in denen die Umwelt dynamisch und nicht vorhersehbar ist.

#### Ausgelagerte Koordination

Alternativ zur nicht-ausgelagerten Koordination beschreibt die Strategie der ausgelagerten Koordination ein Konzept, bei der die Kernfunktionalität von Komponenten separat zur Koordinationsfunktionalität entwickelt wird (vgl. [SR09a]). Dies bedeutet, dass die Modelle der Systemkomponenten parallel zum Prozessmodell, das für die Steuerung der Koordinationsfunktionalität benötigt wird, entwickelt werden. Die Systemkomponenten dienen dabei hauptsächlich als Anbieter von Funktionalitäten, die sie autonom verwalten. Zudem verhalten sich die Komponenten proaktiv, um ihre lokalen Ziele zu erreichen. Die Kommunikation und Koordination, die in diesem Zusammenhang zur Erbringung der angebotenen Funktionalität benötigt wird, lässt sich durch entsprechende Dienste auf Middleware-Ebene bereitstellen. Die Verhaltensmodelle, die der Kommunikation und Koordination zugrunde liegen, sind dabei nicht in dem Modell der Komponenten integriert, sondern in einem separaten Modell ausgelagert und daher unabhängig von den Komponenten und deren Kernfunktionalität. Diese Strategie verfolgt somit auch für den Entwicklungsprozess eine Trennung der Aspekte. Dadurch können die Aspekte der Kern- und der Koordinationsfunktionalität jeweils mit spezialisierten Methoden entwickelt und gewartet werden. Im Bezug auf selbstorganisierende Systeme erhöht der Ansatz allerdings den Aufwand bei der Systementwicklung, da zwei unterschiedliche Typen von Modellen entwickelt und die zugehörigen Implementierungen gepflegt werden müssen. Diesem Nachteil steht der Vorteil der höheren Flexibilität entgegen, der sich dadurch ergibt, dass der Aspekt der Koordination explizit gemacht wird und dadurch besser weiterentwickelt und angepasst werden kann. Dieser Ansatz ist daher insbesondere für Anwendungsdomänen von Interesse, in denen die Koordinationsfunktionalität oft und dynamisch verändert werden muss. Zudem eignet sich der Ansatz für die schnelle Entwicklung von Prototypen und für Situationen, in denen das Verhalten von Komponenten auf Grund hoher Dynamiken zu Beginn des Entwicklungsprozesses nur unzureichend spezifiziert werden kann. Dies ist eine typische Eigenschaft von modernen, verteilten Systemen.

#### Partiell ausgelagerte Koordination

Die Entwicklungsstrategie der partiell ausgelagerten Koordination ist ein hybrider Ansatz, der Konzepte aus den beiden zuvor beschriebenen Strategien verbindet. Diese Strategie zielt dabei auf Situationen ab, in denen die Entwickler zu Beginn der Entwicklung nicht vorhaben, ein System mit selbstorganisierenden Verhalten zu entwickeln. Diese Notwendigkeit ergibt sich dabei erst im Laufe des Entwicklungsprozesses. Somit stehen die Entwickler vor der Herausforderung, die Funktionalität, die zur Erreichung von Selbstorganisation benötigt wird, nachträglich in dem System, welches a priori schon entwickelt und getestet wurde, zu implementieren. Ein möglicher Lösungssatz aus dem *Autonomic Computing* (siehe Abschnitt 2.2.3 und [KC03]) ist das Hinzufügen von *verwaltenden Einheiten* (engl. *„Managing Entities“*). Ein anderer Ansatz besteht darin, die bereits im System vorhandene Dynamik zu analysieren und dann durch das Verändern bzw. Hinzufügen weiterer kausaler Abhängigkeiten die Dynamik mit Hinblick auf die vorgesehenen Systemziele anzupassen. Eine ausführliche Beschreibung dieses Vorgangs findet sich in [SR10a].

#### Zusammenfassung

Allgemein zeigt sich, dass es bei der Entwicklung selbstorganisierender Systeme unterschiedliche Ansätze gibt, um Aspekte der Kern- und der Koordinationsfunktionalität zu modellieren und zu implementieren. Mit Hinblick auf die hohe Dynamik und Unsicherheit in der Umwelt, mit der selbstorganisierende Systeme häufig konfrontiert sind, sind die Strategien der ausgelagerten und partiell ausgelagerten Koordination gut geeignet. Diese beiden Strategien bieten die notwendige Flexibilität, um Systeme im Hinblick auf neue und a priori unbekannte Situationen effizient anzupassen. In Anwendungsgebieten mit relativ einfachen und wenig dynamischen Umgebungen bietet sich hingegen die Strategie der nichtausgelagerten Koordination an, die auf den Mehraufwand, der durch die separate Modellierung und Wartung der Koordinationsfunktionalität entsteht, verzichtet.

### 3.2.3. Existierende Entwicklungsansätze

In diesem Abschnitt werden einige Ansätze diskutiert und bewertet, die das Ziel haben, die systematische Entwicklung selbstorganisierender Systeme zu unterstützen. Dabei lassen sich, unabhängig von der Wahl eines konkreten Softwareentwicklungsmodells, Design- und Implementationsphase als übergeordnete Phasen innerhalb der Modelle unterscheiden. In der Designphase stehen die Aspekte Anforderungsanalyse, Spezifikation, Entwurf und Modellierung des Systems im Vordergrund. In der darauf folgenden Implementationsphase hingegen stehen die konkrete Implementation des Systems und seine Evaluierung im Vordergrund.

### Unified Process:

*Unified Process* (UP) [JBR99] ist ein weit verbreiteter Ansatz für die Entwicklung klassischer Softwaresysteme, der in [DWH05] für die Entwicklung selbstorganisierender Systeme angepasst wird. Bei dem daraus resultierenden Ansatz handelt es sich um einen vollständigen Ansatz, der alle Phasen der Softwareentwicklung beinhaltet. Abbildung 3.5 verdeutlicht den Aufbau des Ansatzes und zeigt seinen iterativen Charakter.

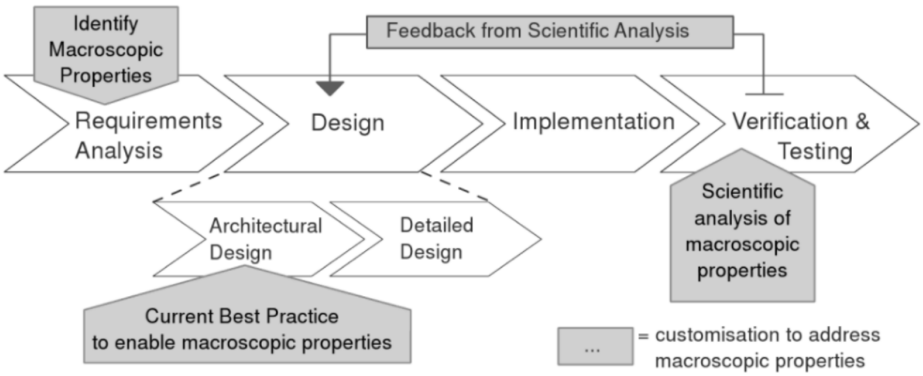



**Abbildung 3.5.:** Anpassung der Entwicklungsmethode Unified Process für selbstorganisierende Systeme - nach (DWH05, S. 4).

Der Ansatz basiert auf dem Gedanken, ein System in Artefakte zu zerlegen und für diese die einzelnen Schritte des Entwicklungszyklus zu durchlaufen. Dadurch ist schon früh möglich, Rückschlüsse über das tatsächliche Systemverhalten zu gewinnen und mögliche Fehlentwicklungen zu kompensieren. Das Ziel dieses Ansatzes liegt darin, die Entwicklung selbstorganisierender Systeme zu systematisieren. Durch die Aneignung eines bereits etablierten Ansatzes soll dabei die Akzeptanz erhöht und die Einarbeitungszeit für Entwickler reduziert werden. Im Kern zielt der Ansatz darauf ab, die ad-hoc Entwicklung selbstorganisierender Systeme durch eine systematische und methodenorientierte Entwicklung zu ersetzen. Insgesamt hat diese Vorgehensweise drei Vorteile, die besonders hervorzuheben sind:

1. Sie basiert auf einem weit verbreiteten Ansatz und versucht so, die Entwicklung selbstorganisierender Systeme auf die Grundlage etablierter Methoden zu setzen.

2. Der Ansatz thematisiert bereits in der frühen Entwicklungsphase die Fragestellung, ob für das vorgesehene Systeme überhaupt selbstorganisierende bzw. emergente Eigenschaften notwendig sind.

3. Der Ansatz hat einen starken Fokus auf der Verifikation des implementierten Systems. Hier wird u.a. eine Methode präsentiert, wie mit der Hilfe von Simulation und Numerik das Verhalten verifiziert werden kann.

Dem entgegen stehen allerdings auch einige Schwächen, die dieser Ansatz hat. Die größte liegt dabei darin, dass der Ansatz insgesamt eher abstrakter Natur und damit in vielen Teilen zu unspezifisch ist. So kann zwar generell überzeugend dargestellt werden, wie der UP auf selbstorganisierende Systeme angewendet werden kann. Der konkreten Anwendung fehlt aber eine Systematik, welche die Verbindung zwischen mikroskopischer und makroskopischer Ebene anleitet. Zudem wird die Frage der konkreten Systemimplementation nicht beantwortet und weder die Beziehung zwischen Kern- und Koordinationsfunktionalität thematisiert noch wird Unterstützung durch ein Framework geboten. Weiterhin fokussiert sich der Ansatz sehr stark auf das Agentenparadigma und unterstützt keine generellen Paradigmen für verteilte Systeme.

#### Konzeptionelles Framework nach Gershenson:

In [Ger06, Ger07] wird ein konzeptionelles Framework für das Design und die Kontrolle in selbstorganisierenden Systemen vorgestellt. Das von *Gershenson* entwickelte Framework gliedert die Systementwicklung dabei in fünf aufeinander folgende Schritte. Dabei nutzt das Framework einen iterativen Ansatz, sodass von jedem Schritt $n$ die Möglichkeit besteht, zum Schritt $n-1$ zurückzuspringen und diesen anzupassen. Die Systementwicklung beginnt dabei mit dem Schritt der *Repräsentation*, hier werden u.a. die Anforderungen und Nebenbedingungen erfasst. Im zweiten Schritt, der *Modellierung*, steht der Aspekt der Kontrolle im Vordergrund. Gershenson betont dabei, dass dies für selbstorganisierende Systeme eine zentrale Herausforderung darstellt. Im nächsten Schritt erfolgt die *Simulation*, hier geht es darum, möglichst früh das zu entstehende Systemverhalten zu beobachten und zu analysieren. Dieses unterscheidet den Ansatz von dem zuvor vorgestellten *Unified Process* Ansatz [DWH05], bei dem Simulation tendenziell erst in späteren Phasen zum Einsatz kommt. Im vierten Schritt erfolgt schließlich die *Implementation* des Systems, ehe es im letzten Schritt zur *Evaluation* kommt. Allerdings ist auch dieser Ansatz eher allgemein und abstrakt gehalten und daher wenig geeignet, die Entwicklung selbstorganisierender Systeme konkret anzuleiten. Nichtsdestotrotz gibt der Ansatz hilfreiche Hinweise darauf, welche Aspekte bei der Entwicklung zu beachten sind.

#### Experimentelle Methode nach Edmonds:

In [Edm05a] wird mit der sogenannten *experimentellen Methode* zwar keine Entwicklungsmethodik im engeren Sinne präsentiert, dafür aber ein grundlegender Paradigmenwechsel propagiert. Ausgehend von den Charakteristika selbstorganisierender Systeme und den damit verbundenen Herausforderungen stellt [Edm05a] fest, dass klassische Ansätze zur Systementwicklung nur bedingt geeignet sind. Solche klassischen Ansätze beruhen prinzipiell darauf, dass basierend auf den Systemanforderungen und den zugehörigen Modellen ein Konstruktionsplan erstellt werden kann, nach dessen Umsetzung zu

großen Teilen sicher ist, dass sich das implementierte System wie ursprünglich vorgesehen verhält. Diese Klasse von Entwicklungsmethoden werden oft als *ingenieurmäßiger* Ansatz beschrieben. Da ein solches Vorgehen für selbstorganisierende Systeme aber nur bedingt sinnvoll ist, wird in [Edm05a] eine experimentelle Methode vorgeschlagen, die den ingenieurmäßigen Ansatz mit einem adaptiven verbindet. Für solche adaptiven Ansätze ist es charakteristisch, dass sie typischerweise zum Einsatz kommen, nachdem ein System konstruiert wurde. Dabei wird das wahrgenommene Verhalten des Systems analysiert und durch die Veränderung von Parametern und Strukturen versucht, das Verhalten des Systems in Richtung des gewünschten Verhaltens zu beeinflussen. Die Kombination dieser beiden Methoden nach [Edm05a] soll die Vorteile der einzelnen Ansätze nutzen, um zielgerichtet selbstorganisierende Systeme zu entwickeln. Verdeutlicht wird dieses Zusammenspiel der beiden Methoden in Abbildung 3.6.

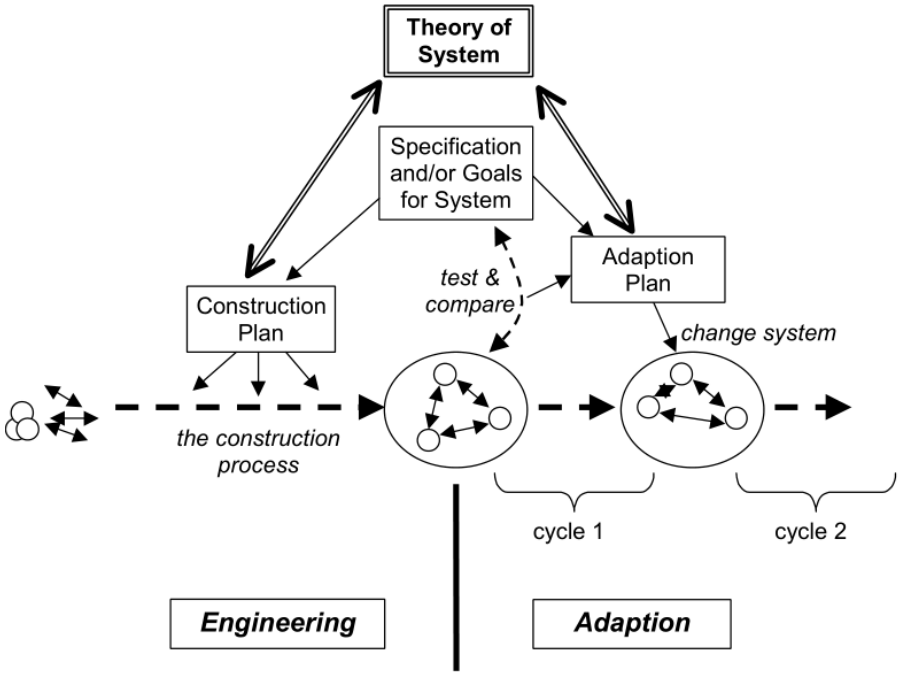



**Abbildung 3.6.:** Experimentelle Methode zur zielgerichteten Entwicklung von selbstorganisierenden Systemen - nach (Edm05a, S. 94).

Im adaptiven Teil werden u.a. Hypothesen aufgestellt und auf der Basis von entsprechenden Experimenten überprüft. Das Ziel dabei ist es, belastbare Aussagen über das tatsächliche Systemverhalten zu gewinnen, die auch Randbedingungen einschließen können. Ein Beispiel für einen auf dem Paradigma der experimentellen Methode basierenden Ansatz wird in [DTc+04] für die Entwicklung sogenannter *Swarm-Bots* präsentiert. Bei diesen handelt es sich um komplexe, selbstorganisierende Roboter, die sich aus einfachen, autonomen Robotern zusammensetzen, die jeweils eine spezielle Fähigkeit besitzen. Innerhalb des Entwicklungsansatzes werden Evolutionsmethoden eingesetzt, um

die Selbstzusammenstellung bzw. das Selbstzusammenfügen der Roboter zu ermöglichen.

#### Agenten-zentrierte Methoden:

Eine besondere, Agenten-zentrierte Methode zur Entwicklung selbstorganisierender Systeme wird in [MMG+09] in Form einer *zielorientierten Methode* präsentiert. Der vorgestellte Ansatz beruht dabei auf der Kombination von zwei etablierten Softwareentwicklungsmethoden für Multiagentensysteme. Dabei kommen *TROPOS* [BPG+04] bzw. dessen Erweiterung für adaptive Systeme *TROPOS4AS* [MPP08] und *ADELFE* [PG04] zum Einsatz. Bei *TROPOS* handelt es sich um eine Methodologie für einen Agenten-orientierten Software Engineering (AOSE) Prozess, der einen Top-down-Ansatz verfolgt, bei dem zunächst die Systemziele auf globaler Ebene definiert werden. In den folgenden Schritten werden diese weiter zerlegt, sodass sie schließlich auf Funktionalitäten auf der Agentenebene abgebildet werden können. In *ADELFE* wird ein Entwicklungsansatz verfolgt, der die lokalen Interaktionen und Kooperationsregeln von Agenten modelliert. Auf Basis der Aktivitäten und Funktionalitäten von Agenten auf der unteren Systemebene entsteht dabei auf globaler Systemebene eine übergeordnete Dynamik, die zu einem emergenten Verhalten führen kann.

*ADELFE* und auch *TROPOS* verwenden dabei Methoden und Konzepte aus anderen Bereichen der Informatik und entwickeln aus deren kombinierter Anwendung neuartige Vorgehensweisen. Dadurch soll die Einarbeitszeit für Entwickler möglichst gering gehalten werden, indem etablierte Techniken zur Anwendung kommen. So basiert z.B. *ADELFE* u.a. auf dem *Rational Unified Process* [JBR99], *UML* und *AUML* [OPB03]. Mit der Kombination dieser beiden Ansätze zu der zielorientierten Methode zielen die Autoren von [MMG+09] darauf ab, die Herausforderungen der systematischen Verbindung zwischen mikroskopischer und makroskopischer Ebene besser adressieren zu können. Die Vorteile dieses Ansatzes liegen in der Nutzung etablierte Softwareentwicklungsmethoden für Multiagentensysteme, allerdings ist die Fähigkeit zur Selbstorganisation dabei nicht wirklich ein zentraler Aspekt der Methode. Der Fokus liegt vielmehr auf Agenten und versucht dann nachträglich, diese so zu modellieren, dass sich ein selbstorganisierendes Verhalten herausbildet. Dabei wird u.a. nicht explizit auf die Beziehung zwischen Kern- und Koordinationsfunktionalität eingegangen. Vielmehr sind beide Aspekte ineinander verwoben, wobei der Fokus eher auf der Modellierung der Kernfunktionalität liegt. Ein weiterer Nachteil ist der starke Fokus auf die Entwurfsphase. Wie die Autoren selber betonen, fehlt dem Ansatz dadurch der wichtige Aspekte der Evaluation des tatsächlichen Systemverhaltens. Wie bereits in [Vil13] festgestellt wurde, steht der Ansatz der zielorientierten Methode stellvertretend für viele (speziell Agenten-zentrierte) Ansätze, die versuchen, etablierte Softwareentwicklungsmethoden für selbstorganisierende Systeme anzupassen. Dies gelingt im konkreten Fall auch gut für die frühen

Entwicklungsphasen, während die späteren Phasen, insbesondere in Form der Konstruktion und Evaluation, nicht abgedeckt werden.

**SodekoVS Ansatz:**

Innerhalb des von der Deutschen Forschungsgesellschaft (DFG) geförderten Forschungsprojekts *„Selbstorganisation durch dezentrale Koordination in Verteilten Systemen“* (SodekoVS) [SBP+09, Sud10] ist ein systematischer Ansatz zur Entwicklung selbstorganisierender Systeme entstanden. Das übergeordnete Ziel von *SodekoVS* war es, einen ganzheitlichen Ansatz zu entwickeln, der sowohl für die Entwurfs- als auch für die Implementierungsphase jeweils Methoden und Werkzeuge bereitstellt, sodass die Modellierung und Konstruktion selbstorganisierender Systeme systematisch angeleitet erfolgen kann. Dabei basiert der SodekoVS-Ansatz auf drei Säulen, die jeweils spezifische Aspekte abdecken. Die erste Säule bezieht sich auf den eigentlichen Entwicklungsprozess. Die SodekoVS-Methodologie beschreibt dabei ein iteratives Vorgehen, das aus fünf Schritten besteht; diese werden in Abbildung 3.7 veranschaulicht. Die Methodik verfolgt einen Top-Down-Ansatz, bei der das gewünschte Systemverhalten in der ersten Phase in den Anforderungen spezifiziert wird (I). Dementsprechend werden in der Analyse-Phase (II) passende Koordinationsmetaphern identifiziert und so kombiniert, dass sie das gewünschte adaptive Verhalten hervorbringen können. In der Design-Phase (III) werden aus diesen Metaphern die entsprechenden Koordinationsstrategien für die Realisierung abgeleitet. Für die einzelnen Koordinationsstrategien existieren generische Implementationen in Software–Bibliotheken, die bei der Realisierung des Systems (IV) verwendet werden können. In der Test-Phase (V) wird das System durch Simulation validiert. Dabei wird das Systemverhalten, das bei der Simulation auftritt, beobachtet und mit dem in den Anforderungen spezifizierten Verhalten verglichen. Bei Abweichungen muss das Design verändert werden und möglicherweise sogar auf andere Metaphern zurückgegriffen werden.

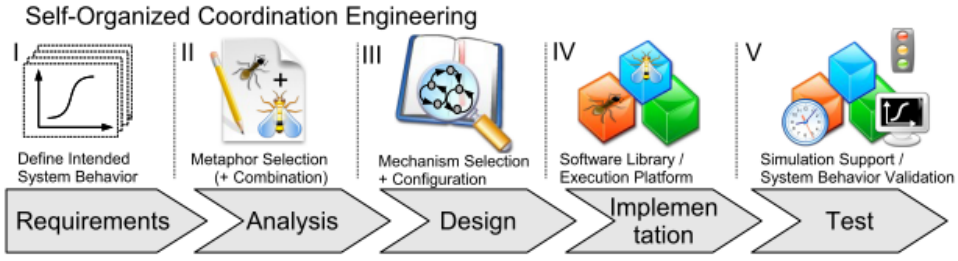



**Abbildung 3.7.:** SodekoVS Vorgehensmodell - nach (SBP+09, S. 7).

Der Ansatz basiert weiterhin auf der Annahme, dass die Kernfunktionalität getrennt von der Koordinationsfunktionalität entwickelt wird. Dabei wird der SodekoVS-Entwicklungsprozess als Ergänzung zum tatsächlichen Softwareentwicklungsprozess verstanden. Daher beschränkt sich der SodekoVS-Entwicklungsprozess auch jeweils auf die Aspekte, die notwendig sind, um

selbstorganisierendes Verhalten durch dezentrale Koordination zu erreichen. Für jeden Entwicklungsschritt wird dabei angeleitet, was speziell getan werden muss, damit das System am Ende auf globaler Ebene ein selbstorganisierendes Verhalten zeigt.

Die zweite Säule des Ansatzes bezieht sich auf die Modellierung. Diese legt im Gegensatz zu den bisher diskutierten Ansätzen den Fokus besonders auf die frühen Phasen der Systementwicklung. Die bisher diskutierten Ansätze versuchen das *beabsichtige Phänomen*, also die zu generierende Struktur zu modellieren, der SodekoVS-Ansatz hingegen schlägt vor, die *Gründe* der selbstorganisierenden Struktur zu modellieren, indem die Feedback-Schleifen modelliert werden, die dem Phänomen zugrunde liegen. Dementsprechend beinhaltet die Modellierung innerhalb des SodekoVS-Ansatzes die Nutzung von CLD-Modellen und der ACBG-Beschreibungssprache (vgl. Abschnitt 3.1.1). Darüber hinaus wird eine Bibliothek angeboten, die etablierte und naturinspirierte Koordinationsmechanismen anbietet und in Bezug auf ihre Charakteristika klassifiziert.

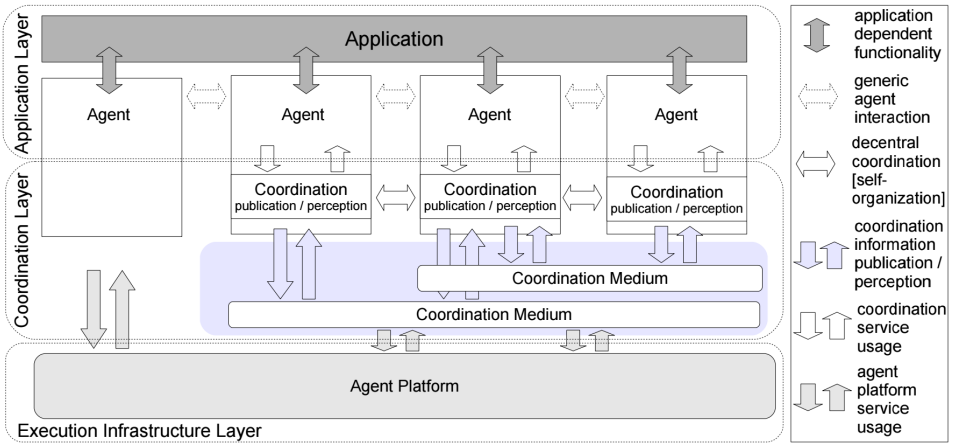



**Abbildung 3.8.:** SodekoVS Referenzarchitektur - nach (SBP+09, S. 6).

Die dritte Säule des Ansatzes beschäftigt sich schließlich mit der Programmierung selbstorganisierender Systeme und fokussiert damit die Implementierungsphase des Entwicklungsprozesses. Im Mittelpunkt dieser Säule steht die SodekoVS-Referenzarchitektur, die in Abbildung 3.8 dargestellt wird. Die Architektur selber besteht aus drei Schichten. In der obersten Schicht befindet sich die eigentliche Anwendungslogik, die hier auf dem Agentenparadigma basiert. Auf der untersten Ebene befindet sich die Ausführungsplattform bzw. die Infrastruktur. Zwischen diesen beiden Schichten befindet sich die *Koordinationsschicht*, die hier im Sinne einer Middleware agiert. Die Aufgabe dieser Schicht ist, die dezentrale Koordinationsfunktionalität zu kapseln, die hier als wesentliches Mittel zur Selbstorganisation genutzt wird. Innerhalb der Koordinationsschicht befinden sich die *Koordinationsmedien*, die primär für den Austausch von Informationen zwischen den Komponenten zuständig sind. Die Komponenten/Agenten selber werden um *Koordinationsendpunkte* (engl. *„Co-*

*ordination Enactment*“ siehe auch Abbildung 3.9) erweitert. Diese Endpunkte sind für die Koordination der Komponenten verantwortlich, indem sie diese beobachten, Informationen über die Koordinationsmedien senden und empfangen und das Verhalten der Komponente anpassen.

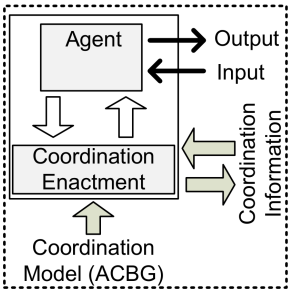



**Abbildung 3.9.:** Funktionsweise des Koordinationsendpunktes im SodekoVS-Ansatz - Ausschnitt aus (VSL+10, Abb. 1, S. 2).

Der im SodekoVS Projekt entwickelte Ansatz beschäftigt sich mit dem kompletten Entwicklungsprozess selbstorganisierender Systeme. Die Stärken des Ansatzes liegen dabei insbesondere in den frühen Phasen der Entwicklung, für die ein neuartiges Paradigma für die Modellierung selbstorganisierender Systeme präsentiert wurde. Durch die vorgestellte Referenzarchitektur werden aber auch die späteren Phasen der Systementwicklung systematisch unterstützt. Allerdings besteht für die Phasen der Implementierung und Evaluierung durchaus Verbesserungspotential. Dabei wäre eine Lösung von der Beschränkung auf das Agentenparadigma wünschenswert, um die Reichweite des Ansatzes zu erhöhen, indem auch andere aktuelle Paradigmen für verteilte Systeme mit einbezogen werden. An dieser Stelle versucht diese Dissertation einen Beitrag zu leisten, indem eine Lösung präsentiert wird, die den SodekoVS Ansatz aufgreift und ihn so generalisiert, dass dieser verteilte Systeme im Allgemeinen und nicht nur Multiagentensysteme im Speziellen unterstützt (siehe Kapitel 6). Im Bezug auf die Evaluierung wären z.B. Metriken sinnvoll, die Auskunft über die Güte bzw. die Qualität des Systems geben, insbesondere in Bezug auf die nichtfunktionalen Anforderungen, die ein Kernelement selbstorganisierender Systeme sind. An dieser Stelle leistet die Dissertation von *Vilenica* [Vil13] einen wertvollen Beitrag, die den SodekoVS-Ansatz um ein Rahmenwerk erweitert, das die Evaluierung durch die automatisierte Ausführung von Systemsimulationen und deren Auswertung im Bezug auf vorher definierte Kennwerte erweitert.

### Fazit

Abschließend lässt sich feststellen, dass die vorausgegangene Betrachtung von Ansätzen zur systematischen Entwicklung von selbstorganisierenden Systemen zeigt, dass die einzelnen Ansätze sehr unterschiedliche Strategien verfolgen. So wird beispielsweise versucht, etablierte Entwicklungsmethoden aus der Softwareentwicklung auf selbstorganisierende Systeme zu übertragen

oder es werden existierende Ansätze, speziell für die Entwicklung von Multiagentensystemen, kombiniert. Gemeinsam haben alle Ansätze dabei, dass betont wird, dass sich die Entwicklung selbstorganisierende Systeme grundsätzlich von klassischen verteilten Systemen unterscheidet. So spielen z.B. Aspekte wie Systemsimulation und Evaluierung eine besondere Rolle. Dies wird am deutlichsten am Ansatz der experimentellen Methode nach *Edmonds* [Edm05a]. Eine weitere Gemeinsamkeit der untersuchten Ansätze liegt darin, dass sie im Allgemeinen einen relativ starken Fokus auf die Designphase legen und den Entwicklern für die spätere Phase der Implementierung wenig Anleitung geben. Dieses wird u.a. dadurch deutlich, dass es kaum konkrete Empfehlungen gibt, wie die Beziehung zwischen Koordinations- und Kernfunktionalität zu gestalten ist. Allgemein birgt eine Vernachlässigung der Implementationsphase die Gefahr, dass Systeme konstruiert werden, die in Bezug auf Prinzipien der Softwaretechnik nicht optimal sind und ggf. Insellösungen produzieren, die nicht allgemein nutzbar sind. An dieser Stelle hebt sich lediglich der SodekoVS-Ansatz [SBP+09] hervor, der explizit die systematische Konstruktion selbstorganisierender Systeme mit seiner Referenzarchitektur berücksichtigt. Der Aspekt der Evaluierung wird in den betrachteten Ansätzen ebenfalls nur unzureichend behandelt. Eine Ausnahme bildet hier wieder der SodekoVS-Ansatz, für den in der Dissertation von *Vilenica* [Vil13] ein Rahmenwerk konstruiert wurde, das die automatische Evaluierung des implementierten Systems auf Basis von Systemsimulationen unterstützt.

## 3.3. Zusammenfassung

In diesem Kapitel wurde die systematische Entwicklung selbstorganisierender Systeme betrachtet. Dabei wurden zunächst die Hauptbestandteile selbstorganisierender Systeme identifiziert und im Einzelnen betrachtet. Dadurch konnte die Funktionsweise selbstorganisierender Systeme dargelegt und insbesondere deutlich gemacht werden, welche Aspekte es bei der systematischen Entwicklung zu bedenken gibt. Bezüglich der Hauptkomponenten selbstorganisierender Systeme, die in diesem Kapitel identifiziert wurden, wird dabei u.a. deutlich, dass Feedbackschleifen essentiell sind, wenn es darum geht, dynamisches und adaptives Verhalten in selbstorganisierenden System zu realisieren. Wenn es um die Umsetzung der eigentlichen Kernfunktionalität selbstorganisierender Anwendungen geht, so sind Agenten bzw. Aktive Komponenten besonders geeignet, da sie inhärent eine Aufteilung der Anwendungslogik in einzelne, autonome Funktionseinheiten unterstützen. Diese Aufteilung der Funktionalität in mehrere Komponenten bedingt die Nutzung dezentraler Koordination, um die Abhängigkeiten zwischen den verteilten Entitäten einer Anwendung verwalten zu können. Dabei kann die Umgebung der Anwendung u.a. dazu genutzt werden, um indirekte Formen der Kommunikation in selbstorganisierenden Systemen zu realisieren.

Darauf aufbauend wurden die speziellen Herausforderungen untersucht, die die systematische Entwicklung selbstorganisierender Systeme mit sich bringt.

Hier wurde u.a. die Abbildung der Beziehung zwischen den Systemparametern auf mikroskopischer Ebene und der tatsächlich beobachteten Dynamik auf makroskopischer Ebene als herausfordernd identifiziert. Ferner können das nichtdeterministische und nichtlineare Verhalten selbstorganisierender Systeme zu unerwünschten, emergenten Eigenschaften bzw. Phänomenen führen. Außerdem wurde herausgearbeitet, dass selbstorganisierende Systeme formal nur schwer zu analysieren sind und daher neuartige Ansätze nötig sind, um das Verhalten und die Qualität dieser Art von Systeme zu evaluieren.

Folglich wurde eine Reihe existierender Ansätze, die sich mit der systematischen Entwicklung von selbstorganisierenden Systemen beschäftigten, vorgestellt und analysiert. Dabei wurde deutlich, dass die Mehrheit der untersuchten Ansätze einen relativ starken Fokus auf die früheren Phasen, insbesondere die Designphase, legen und wenig Anleitung für die spätere Implementierung geben. Dieses wird vor allem dadurch deutlich, dass sie kaum konkrete Empfehlungen geben, wie die Beziehung zwischen Kern- und Koordinationsfunktionalität zu gestalten ist. Von den untersuchten Ansätzen hebt sich lediglich der *SodekoVS*-Ansatz hervor, der explizit die systematische Konstruktion selbstorganisierender Systeme mit einer Referenzarchitektur adressiert.

Zusammengefasst verdeutlicht dieses Kapitel, was es bei der systematischen Entwicklung selbstorganisierender Systeme zu bedenken gilt, damit die Vorteile des Paradigmas der Selbstorganisation für moderne, verteilte Systeme nutzbar gemacht werden können.

# 4. Konstruktion selbstorganisierender Multiagentensysteme

Die Entwicklung selbstorganisierender Systeme durchläuft, beginnend mit der Konzeption, unterschiedliche Phasen und endet nach Analyse- und Designphase schließlich mit der Realisierung bzw. der Evaluation des Systems. In diesem Kapitel liegt der Fokus auf dem Schritt der Konstruktion, also der tatsächlichen Realisierung und Implementierung. Dabei wird zunächst herausgearbeitet, welche speziellen Herausforderungen für die Konstruktion selbstorganisierender Systeme bestehen. Anschließend wird ein spezieller Ansatz vorgestellt, der darauf abzielt, diese Herausforderungen zu bewältigen und dabei eine Lösung für die Konstruktion selbstorganisierender Multi-Agenten Systeme liefert. Aufbauend auf dieser Lösung beschäftigt sich das Kapitel anschließend mit der strukturellen Adaption von Koordinationsprozessen. Dabei geht es um den dynamischen Austausch bzw. die Umkonfiguration der Koordinationsprozesse, die das selbstorganisierende Verhalten der Anwendung abbilden, wenn sich zur Laufzeit Bedingungen ergeben, unter denen die selbstorganisierende Dynamik zu unerwünschten bzw. nicht dem gewünschten Verhalten führt.

## 4.1. Spezielle Herausforderungen für die Konstruktion

Für die Konstruktion selbstorganisierender Systeme im Allgemeinen ergeben sich eine Reihe von speziellen Herausforderungen, die im Folgenden behandelt werden. Innerhalb des Entwicklungsprozesses findet die eigentliche Systemkonstruktion zeitlich gesehen in den späteren Phasen statt (vgl. Abschnitt 3.2). Dabei sind die Analyse- und Modellierungsphasen (meistens) bereits abgeschlossen. Während klassische Entwicklungsmethoden die einzelnen Phasen typischerweise eher sequentiell abhandeln (wie z.B. im *Wasserfallmodell* [Roy87]), betonen modernere Entwicklungsmethoden ein iteratives Vorgehen (beispielsweise *Extreme Programming* [BF01] oder *Scrum* [Coh09]), bei denen die einzelnen Phasen wiederholt durchlaufen werden [PP04]. Daher sollte auch die Konstruktion selbstorganisierender Systeme so ausgelegt sein, dass ein iteratives Vorgehen unterstützt wird. Agenten sind hierfür gut geeignet, da sie neben der Fähigkeit autonom zu handeln auch die Möglichkeit bieten, die Gesamtfunktionalität eines Systems in dedizierte Komponenten aufzuteilen. Daher ist es möglich, einzelne Teile des Systems schnell und nebenläufig zu konstruieren, sodass im Sinne der schnellen Entwicklung von Prototypen (engl. *„rapid prototyping“*) [TB90] früh Ergebnisse erzielt und überprüft werden können. Diese können dann ggf. dazu führen, dass das Systemmodel angepasst werden muss. Ein solches Vorgehen ist für die Ent-

wicklung selbstorganisierender Systeme u.a. aufgrund der Herausforderung, die aus der Beziehung zwischen der mikroskopischen und makroskopischen Ebene und möglicher nicht-vorgesehener, emergenter Phänomene entsteht, wünschenswert.

Eine weitere Herausforderung resultiert aus der Beziehung zwischen Kern- und Koordinationsfunktionalität. Die Aufgabe letzterer ist es, die Dynamik bzw. Struktur im System zu erzeugen, die zu dem selbstorganisierenden Verhalten führt. Daher ist die Beziehung zwischen diesen beiden Aspekten von entscheidender Bedeutung. Die Fragestellungen, die aus dieser Beziehung resultieren, lassen sich übergeordnet als Herausforderungen der Integration ansehen. Die Integration zielt darauf ab, einen Ansatz zu finden, der auf der einen Seite die Besonderheiten bei der Konstruktion von Kern- und Koordinationsfunktionalität bedenkt und dabei auf der anderen Seite eine möglichst, kohärente, nebenläufige und effiziente Implementierung ermöglicht. Dabei gilt es zu bedenken, dass die Konstruktion selbstorganisierender Systeme i.d.R. zu einem großen Anteil aus konventioneller Kernfunktionalität besteht und die Koordinationsfunktionalität nur einen verhältnismäßig geringen Anteil ausmacht. Hier stellt sich u.a. die Frage, wie Komponenten mit bzw. ohne Koordinationsfunktionalität konstruiert werden können und wie insbesondere letzterer Aspekt dabei transparent gehalten werden kann. Auch sollte aufgrund des hohen Anteils konventioneller Kernfunktionalität die Konstruktion dieser Komponenten auf etablierten Paradigmen, Technologien, Programmiersprachen und Entwicklungsumgebungen beruhen, um die Einarbeitungszeit für die Entwickler möglichst gering zu halten. Soweit möglich und sinnvoll, sollte dieses auch für die Konstruktion der Koordinationsfunktionalität gelten. Um die hier spezielle Herausforderung der Integration von Kern- und Koordinationsfunktionalität methodisch zu unterstützen, gilt es eine entsprechende Architektur zu finden, die dieses soweit wie möglich unterstützt.

Neben der methodischen Ebene gilt es auch, die Integration für die konkrete Implementierung eines selbstorganisierenden Systems anzuleiten. Hier werden klare Schnittstellenkonzepte, z.B. aber nicht ausschließlich in Form einer Ausführungsumgebung, benötigt, die auf programmiertechnischer Ebene die oben angesprochenen Herausforderungen adressieren, d.h. u.a. die Möglichkeit bieten, Kern- und Koordinationsfunktionalität getrennt voneinander und transparent zu implementieren. Eine weitere zentrale Anforderung liegt darin, die Implementierung der Koordinationsfunktionalität möglichst nahtlos auf Basis der Schnittstellenkonzepte zu ermöglichen. Zusätzlich sollten die Schnittstellen sich an etablierten Prinzipien der Softwareentwicklung wie beispielsweise Wiederverwendbarkeit, Transparenz und loser Kopplung orientieren. Dies ist nicht zuletzt für die Implementierung und Nutzung von Koordinationsmechanismen von entscheidender Bedeutung.

Eine weitere spezifische Herausforderung liegt in der expliziten Adressierung von Umgebungen (vgl. Abschnitt 3.1.4), die einem wichtigen Bestandteil selbstorganisierender Systeme darstellen. Damit soll es ermöglicht werden, auch selbstorganisierende Systeme, bei denen Teile der

Koordinationsfunktionalität in der Umgebung angesiedelt sind, systematisch zu konstruieren.

Des Weiteren liegt eine Herausforderung darin, einen domänenunabhängigen Ansatz zur Konstruktion selbstorganisierender Systeme zu entwickeln. Domänenunabhängigkeit zielt herbei auf zwei Dimensionen ab: Einerseits kann sie als Anwendungsunabhängigkeit verstanden werden, andererseits als Unabhängigkeit von speziellen Formen der Selbstadaptivität, wie sie sich z.B. in den einzelnen Formen der Self-* Eigenschaften (vgl. Abschnitt 2.2.2) äußern. Der Ansatz soll also allgemein genug sein, damit diese beiden Dimensionen unterstützt werden. Gleichzeitig soll es die Möglichkeit geben, durch entsprechende Verfeinerungen Varianten zu konstruieren, die ausgehend vom allgemeinen Ansatz speziell auf gewisse Fragestellungen zugeschnitten sind und diese besonders gut unterstützen.

Zusammenfassend zeigt sich, dass die Konstruktion selbstorganisierender Systeme systematisch angeleitet werden muss. Es wird ein Ansatz benötigt, der dies auf methodischer Ebene durch eine entsprechende Integrationsarchitektur unterstützt und für die Ebene der konkreten Implementierung ein dazu korrespondierendes Schnittstellenkonzept bereitstellt. Zusammen systematisieren sie die Konstruktion, sodass die entwickelten selbstorganisierenden Systeme auf einem einheitlichen Paradigma basieren, die benutzten Selbstorganisationskomponenten wiederverwendbar und transparent sowie die Kern- und Selbstorganisationsfunktionalität klar voneinander getrennt und lose gekoppelt sind.

## 4.2. Vergleich existierender Ansätze

In diesem Abschnitt werden Arbeiten vorgestellt, die sich mit der systematischen Konstruktion von selbstorganisierenden Systemen beschäftigen. Dazu wird zunächst eine grundlegende Klassifikation vorgenommen, ehe auf dieser Basis die existierenden Ansätze diskutiert und bewertet werden.

### 4.2.1. Klassifikation

Betrachtet man die Überlegungen, die im vorangestellten Abschnitt diskutiert wurden, so wird deutlich, dass für die Konstruktion selbstorganisierender Systeme das Agentenparadigma besonders geeignet ist. Zudem wird deutlich, dass der Aspekt der Koordination eine wichtige Rolle bei der Realisierung von selbstorganisierenden Verhalten spielt (vgl. Abschnitt 3.1.2) und hierbei auch die Einbindung von Umgebungen möglich sein sollte. Basierend auf diesen Erkenntnissen wird im Folgenden eine Klassifikation existierender Ansätze vorgenommen, die sich an drei unterschiedlichen Klassen orientiert, die in diesem Kontext in [VSL+10] identifiziert wurden:

1. Integrierte Agentenumgebungen
2. Generische Umgebungsinfrastrukturen

3. Spezialisierte Koordinationsrahmenwerke

#### Integrierte Agentenumgebungen:

Die Klasse der integrierten Agentenumgebungen umfasst Ansätze, bei denen Umgebungen ein integraler Bestandteil des Multiagentensystems sind. Hierbei findet allerdings eine sehr enge Kopplung dieser beiden Aspekte statt, daher ist eine getrennte Konzeption und Implementierung von Kern- und Koordinationsfunktionalität kaum möglich. Diese Ausprägung findet sich sehr häufig bei Multiagenten-Rahmenwerken, die speziell für Simulationen entwickelt wurden. Beispiele für solche Rahmenwerke sind z.B. *Agent.GUI* [DBU11], *NetLogo* [TW04] und *SimSeSAm* [Klü01]. Diese Rahmenwerke bieten i.d.R. die Möglichkeit, neben den eigentlichen Agenten auch eine Umgebung zu gestalten und dort Komponenten zu platzieren. Im Sinne der indirekten Interaktionen (siehe Abschnitt 2.3.2) können diese Komponenten dazu genutzt werden, selbstorganisierendes Verhalten im System zu realisieren. Die Rahmenwerke bieten dabei oftmals Funktionalitäten an, die es den Agenten erlauben, ihre Umgebung wahrzunehmen und deren Zustand zu beeinflussen. Zudem besteht oft die Möglichkeit, spezielle Umweltprozesse zu spezifizieren, die es ermöglichen, losgelöst von den Agenten separate Funktionslogik zu definieren, die den Zustand der Umwelt verändern kann.

Insgesamt bieten die Ansätze dieser Klasse viele vordefinierte Möglichkeiten, die es prinzipiell ermöglichen, selbstorganisierende Systeme zu konstruieren. Allerdings fehlt es ihnen an Möglichkeiten, die vordefinierten Funktionalitäten für individuelle Anforderungen anzupassen oder durch eigene Komponenten zu erweitern. Aus konzeptioneller Sicht ist zudem die enge Kopplung zwischen Kern- und Koordinationsfunktionalität von Nachteil, da diese u.a. zur Folge hat, dass die Implementation von Agenten angepasst werden muss, wenn sich etwas an der Koordinationsfunktionalität ändert. Hier fehlt es also an der bereits angesprochenen Transparenz zwischen Kern- und Koordinationsfunktionalität. Darüber hinaus eignen sich viele Simulationsrahmenwerke nicht oder nur bedingt dazu, tatsächliche Anwendungen zu konstruieren. Dementsprechend enthält die hier diskutierte Klasse interessante Aspekte, ist aber insgesamt nicht in der Lage, die Konstruktion selbstorganisierender Systeme ausreichend (i.S.v Abschnitt 4.1) zu unterstützen und zu ermöglichen (vgl. [Vil13]).

#### Generische Umgebungsinfrastrukturen:

Die Klasse der generischen Umgebungsinfrastrukturen beinhaltet Ansätze, bei denen Umgebungen ebenfalls ein integraler Bestandteil des eigentlichen Multiagentensystems sind. Im Gegensatz zur vorherigen Klasse unterstützen diese Ansätze allerdings eine umfassendere Modellierung und Konstruktion der Umgebungen. Diese Ansätze bieten also grundlegende Funktionalitäten an, die leicht im Hinblick auf individuelle Anforderungen angepasst und erweitert werden können. Zur Klasse der generischen Umgebungsinfrastruk-

turen zählen u.a. *Agents & Artifacts* [RVO08], *Environment Support* [JBP10] und *MOISE+* [HSB07]. *Agents & Artifacts* nutzt beispielsweise sogenannte Artefakte, die Elemente in der Umgebung darstellen, welche von Agenten verwendet werden können. Die Autoren von [RVO08] diskutieren dabei auch die Möglichkeit, spezielle Artefakte zu entwickeln, die zur Koordination verwendet werden können. Diese werden folglich als *Coordination Artifacts* [ORV+04, RV05] bezeichnet. Damit können bekannte Koordinationskonzepte wie Black-Boards, Karten oder auch Aufgabenplaner realisiert werden. Allerdings müssen die Agenten bei diesem Ansatz die Artefakte explizit zur Koordination nutzen. Damit wird der Aspekt der Koordination zu einem integralen Bestandteil des funktionalen Entwurfs der Agenten. Hierbei findet also auch eine enge Kopplung zwischen der Kern- und Koordinationsfunktionalität statt.

Insgesamt lässt sich feststellen, dass die Klasse der generischen Umgebungsinfrastrukturen Konzepte bereithält, die unter Umständen dazu geeignet sind, um darauf aufbauend selbstorganisierende Systeme zu konstruieren. Die Konzepte bieten grundsätzlich die Möglichkeit, Umgebungen so zu gestalten, dass sie zur Koordination von Agenten verwendet werden können. Während die Flexibilität und Erweiterbarkeit dieser Ansätze dabei positiv zu bewerten sind, ist die fehlende Trennung von Kern- und Koordinationsfunktionalität erneut negativ anzumerken.

#### Spezialisierte Koordinationsrahmenwerke:

Die Klasse der spezialisierten Koordinationsrahmenwerke umfasst Ansätze, die oftmals genutzt werden, um in einem bestimmten Anwendungskontext eine Form der Koordination bereitzustellen. Beispiele für solche Ansätze sind *2APL* [Das08], *Anthill* [BMM02] oder auch *Field-Based Coordination* [MZ06a]. Diese Ansätze eignen sich besonders, wenn nur bestimmte Koordinationsmechanismen benötigt werden. Hier bieten sie eine gute Unterstützung für die Realisierung und Nutzung selbiger. Die Unterstützung reicht dabei, abhängig vom einzelnen Ansatz, von Simulationsumgebungen bis hin zu Rahmenwerken zur Konstruktion der eigentlichen Systeme. Da es bei diesen Ansätzen darum geht, in einem bestimmten Anwendungskontext Koordination bereitzustellen, spielt eine klare Trennung zwischen Kern- und Koordinationsfunktionalität meist keine entscheidende Rolle, da die Koordination i.d.R. Teil der Anwendungslogik ist.

Insgesamt zeigt sich mit der Klasse der spezialisierten Koordinationsrahmenwerke eine weitere Facette, wie selbstorganisierende Systeme konstruiert werden können. Diese Ansätze eignen sich besonders für Anwendungsszenarien, bei denen a priori bekannt ist, welcher Typ von Koordinationsmechanismen benötigt wird. Ist dieses allerdings nicht Fall, so sind die Ansätze dieser Klasse wegen ihrer fehlenden Flexibilität nicht geeignet. Insgesamt gesehen sind sie nicht in der Lage, in generischer Art und Weise die Konstruktion selbstorganisierender Systeme zu ermöglichen. Ihre Nutzbarkeit beschränkt sich daher auf die Unterstützung von speziellen Anwendungsfällen.

### 4.2.2. Bewertung

Abschließend lässt sich nach Analyse der existierenden Ansätze auf dem Gebiet der Konstruktion selbstorganisierender Systeme feststellen, dass es eine Vielfalt von unterschiedlichen Ansätzen gibt. Die Klassifikation nach [VSL+10] ordnet die existierenden Arbeiten dabei ein und macht ihre jeweiligen Stärken und Schwächen deutlich. Dadurch lässt sich auf abstrakter Ebene deutlich machen, welche Art von Ansätzen für welche Anwendungsdomäne bzw. Fragestellungen besonders geeignet sind.

So zeigt die Klasse der integrierten Agentenumgebungen ihre Stärke bei Anwendungsszenarien, bei denen die Bereitstellung der Standardfunktionalität der Ansätze ausreichend ist und keine weiteren, individuellen Erweiterungen nötig sind. Solche Ansätze finden sich häufig im Bereich der Simulationsrahmenwerke. Nachteilig an den Ansätzen dieser Klasse ist daher folglich die fehlende Flexibilität und Erweiterbarkeit in Bezug auf die Konstruktion von Kernfunktionalität sowie die enge Kopplung zwischen dieser und der Koordinationsfunktionalität. Einige dieser Nachteile lassen sich allerdings mit Blick auf die Klasse der generischen Umgebungsinfrastrukturen überbrücken. Dies gilt besonders für die Flexibilität und die Wiederverwendbarkeit, was die charakteristischen Merkmale dieser Klasse sind. Trotzdem bleibt auch hier die mangelnde Unterstützung bzgl. der Trennung von Kern- und Koordinationsfunktionalität ein Problem. Bei der Klasse der spezialisierten Koordinationsrahmenwerke steht die Unterstützung einzelner Koordinationsmechanismen im Vordergrund. Diese Ansätze eignen sich daher besonders gut für Anwendungsszenarien, bei denen nur eine bestimmte Art von Koordination benötigt wird.

Daher wird deutlich, dass keine der beschriebenen Klasse in der Lage ist, die Konstruktion selbstorganisierender Systeme umfassend zu unterstützen und dabei die in Abschnitt 4.1 identifizierten Herausforderungen zu adressieren. Das größte, klassenübergreifende Problem hierbei ist die enge Kopplung von Kern- und Koordinationsfunktionalität. Hier fehlt ein Ansatz, der identifizierten Herausforderungen sowohl auf konzeptioneller als auch implementationstechnischer Ebene systematisch umsetzt. Insbesondere die transparente und nichtinvasive Einbindung in eine funktional umfangreiche Ausführungsumgebung wird von den existierenden Ansätzen so nicht geleistet. Trotzdem zeigt sich, dass die Einbindung bzw. Weiterentwicklung existierende Ansätze gewinnbringend sein kann, um einzelne Teilaspekte zu realisieren. So ist die Flexibilität und Wiederverwendbarkeit der Klasse der generischen Umgebungsinfrastrukturen interessant, genauso wie die gute Unterstützung einzelner Koordinationsmechanismen aus der Klasse der spezialisierten Koordinationsrahmenwerke.

## 4.3. Konstruktionsansatz Coordination Space

Ausgehend von der Analyse existierender Ansätze im vorangegangenen Abschnitt und der dort im Hinblick auf die Ziele dieser Arbeit identifizierten Unzulänglichkeiten wird im Folgenden das im Rahmen der Dissertation von *Vilenica* [Vil13] entwickelte Konzept des *Coordination Space* vorgestellt. Dieses Konzept basiert auf der *SodekoVS*-Referenzarchitektur [SBP+09] und verbindet es mit der generischen Umgebungsinfrastruktur *Environment Support* [JBP10]. Das Konzept des *Environment Support* ermöglicht es, die Umgebung in einem Multiagentensystem dediziert zu modellieren und zu implementieren. Umgesetzt wurde dieses Konzept für die *Jadex*-Agentenplattform [BP12a]. Dementsprechend realisiert der Konstruktionsansatz des *Coordination Space* neben einem konzeptionellen Integrationsmodell auch eine Ausführungsumgebung auf Basis der *Jadex*-Agentenplattform. Durch die Kombination der *SodekoVS*-Referenzarchitektur und des *Environment Support* Ansatzes entsteht ein Konzept, in dem dezentrale Koordination ein Modellierungs- und Implementationsobjekt erster Ordnung ist. Damit wird die Konzeption und Implementation dezentraler, selbstorganisierender Anwendungen systematisiert und unterstützt. Eine weitere wichtige Eigenschaft des *Coordination Space* Ansatzes ist, dass die strikte Trennung von Kern- und Koordinationsfunktionalität unterstützt wird.

Das *Coordination Space* Konzept wurde im Rahmen dieser Dissertation um ein Verteilungskonzept erweitert, das die Nutzung von Koordinationsräumen über mehrere Plattformen und Netzwerkknoten hinweg ermöglicht [PVR13]. Basierend auf dieser Erweiterung wurde schließlich, ebenfalls im Rahmen dieser Dissertation, ein Konzept entwickelt, das die strukturelle Adaption von Koordinationsprozessen zur Laufzeit und damit den Austausch und die Rekonfiguration dieser Prozesse unterstützt [PR15, PRD15] (siehe Abschnitt 4.4).

Im Folgenden wird zunächst die Integrationsarchitektur des *Coordination Space* Ansatzes beschrieben, bevor die auf dem *Jadex*-Rahmenwerk basierende Ausführungsumgebung vorgestellt wird. Darauf aufbauend wird dann das im Rahmen dieser Arbeit entwickelte Verteilungskonzept für Koordinationsräume vorgestellt.

### 4.3.1. Integrationsarchitektur

Das konzeptionelle Modell des *Coordination Space* ist in Abbildung 4.1 dargestellt. Es zeigt, dass der *Coordination Space* ein Teil der Umgebung ist, in der eine Softwarekomponente, z.B. ein Agent, situiert ist. Dabei kapselt der *Coordination Space* alle Koordinationsaspekte der Anwendung und interagiert transparent mit denjenigen Komponenten der Anwendung, die koordiniert werden sollen. Die wichtigsten Komponenten, aus denen sich das konzeptionelle Modell des *Coordination Space* zusammensetzt, sind dabei:

**Agent:** Der Agent ist eine der Komponenten, welche die eigentliche Kernfunktionalität der Anwendung realisieren und die koordiniert werden sollen,

um das vorgesehene selbstorganisierende Verhalten in der Anwendung umzusetzen.

**Listener/Observer:** Sind spezielle Komponenten, welche die Zustände der Agenten im Hinblick auf solche, die für das Koordinationsmodell relevant sind, überwachen. Ihre Aufgabe ist es, spezielle Koordinationsereignisse zu generieren sobald ein solcher relevanter Zustand auftritt.

**Perceive-Schnittstelle:** Die *Perceive*-Schnittstelle ist eine asynchrone Schnittstelle, über die der *Coordination Space* (Koordinations-) Ereignisse wahrnehmen kann. Die Schnittstelle ist dabei asynchron gestaltet, was für verteilte, dynamische Systeme von Vorteil ist.

**Koordinationsmedium:** Das Koordinationsmedium stellt die Kernkomponente der Koordinationsfunktionalität dar. Seine Aufgabe ist es, ankommende Koordinationsereignisse entsprechend des vorgegebenen Koordinationsmodells zu verarbeiten und zu verbreiten.

**Publish-Schnittstelle:** Die *Publish*-Schnittstelle ist eine spezielle Schnittstelle des *Coordination Space*, die es Komponenten erlaubt, dort Ereignisse zu publizieren.

**Percept-Generator:** Die *Percept*-Generatoren sind Komponenten im *Coordination Space*, die dort publizierte Ereignisse verarbeiten können, um daraus spezifische Empfindungen (Engl. *„perception“*) zu erzeugen, die wiederum von den Agenten, die ebenfalls im Raum situiert sind, wahrgenommen werden können. Ein *Percept*-Generator ist dabei immer für eine bestimmte Agenten-Architektur zuständig.

**Percept-Prozessor:** *Percept*-Prozessoren sind Teil der Agenten und dafür zuständig erzeugte *perceptions* wahrzunehmen und weiter zu verarbeiten.

Der funktionale Aufbau des *Coordination Space* lässt sich am besten anhand des Ablaufs der erfolgenden Koordination beschreiben (vgl. [Vil13]). Hierbei wird zunächst die Komponente *Init Agent Listeners and Observe* betrachtet (siehe Abbildung 4.1). Zu Beginn der Koordination ist es die Aufgabe dieser Komponente, die notwendigen *Listener* zu initialisieren, um die Agenten zu überwachen. Die Initialisierung der *Listener* orientiert sich dabei an dem zugrundelegenden Koordinationsmodell, das beschreibt, welche Eigenschaften des Agenten zu beobachten sind. Der Ansatz der *Listener* setzt dabei voraus, dass die zu beobachtenden Agenten über Schnittstellen verfügen, die eine Observierung ihrer internen Zustände durch externe Komponenten ermöglichen. Als Koordinationsmodell wird dabei das *MASDynamics*-Modell [SR09b] (vgl. Abschnitt 3.1.2) genutzt. Besonders hervorzuheben sind dabei die deklarativen Beschreibungsansätze, die sich darauf konzentrieren, *was* koordiniert werden soll und nicht *wie*.

Nachdem die Initialisierungsphase abgeschlossen ist beobachten die *Listener* die Agenten. Sie werden aktiv, sobald im Agenten Zustände oder Ereignisse

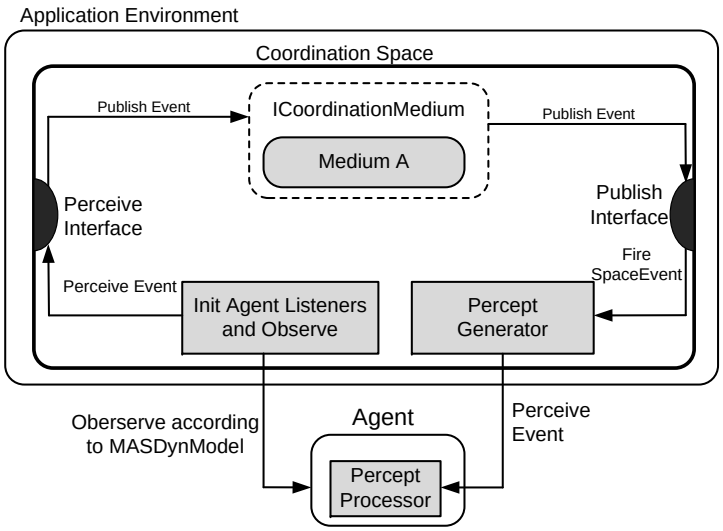



**Abbildung 4.1.:** Konzeptionelles Modell des *Coordination Space* - aus (VSL+10).

beobachtet werden, welche im Koordinationsmodell spezifiziert wurden. Bei Agenten, die auf der BDI-Architektur (siehe [RG+95]) basieren, kann dieses z.B. der Start eines Zieles, das Auftreten eines internen Ereignisses oder auch die Wertveränderung eines *Beliefs* sein. Unabhängig von der Art der Zustandsänderung verarbeitet der *Listener* diesen, sofern er gemäß Koordinationsmodell relevant ist. Dazu generiert der *Listener* ein Koordinationsereignis, das neben der eigentlichen Beobachtung noch weitere Verwaltungsinformationen, die zur Verarbeitung benötigt werden, enthält.

Das generierte Koordinationsereignis wird anschließend vom *Listener* über die *Perceive*-Schnittstelle an den *Coordination Space* weitergereicht, sodass dieser das Ereignis wahrnehmen kann. Diese Schnittstelle ermöglicht somit eine lose Kopplung zwischen den Komponenten, die Koordinationsereignisse erzeugen und dem *Coordination Space*, der diese weiterverarbeitet.

Nachdem der *Coordination Space* das Koordinationsereignis empfangen hat, wird auf Basis der enthaltenen Verwaltungsinformationen entschieden, an welches Koordinationsmedium das Ereignis weitergeleitet wird.

Das Koordinationsmedium verarbeitet nun das erhaltene Koordinationsereignis gemäß dem Koordinationsmodell und nutzt dann die *Publish*-Schnittstelle des *Coordination Space*, um dort wiederum die Ergebnisse der Verarbeitung zu publizieren. Dabei ist die Bandbreite der möglichen Semantik und damit auch der dadurch realisierbaren Funktionalität im jeweiligen Koordinationsmedium sehr breit. Auf der einen Seite stehen relativ einfache Varianten, bei denen das Koordinationsmedium lediglich dazu genutzt wird, um Koordinationsereignisse zwischen den Agenten zu kommunizieren. In diesem Fall werden die Koordinationsereignisse vom Koordinationsmedium selber i.d.R. nicht verändert. Auf der anderen Seite stehen Varianten, bei denen das Koordinationsmedium selber Funktionalität besitzt und eigene Berechnungsschritte durchführt. In diesem Fall nimmt es eine aktive Rolle

ein und ist nicht, wie im ersten Fall dargestellt, nur darauf beschränkt, als Vermittler von Nachrichten zu dienen. Aktive Koordinationsmedien eignen sich beispielsweise für Anwendungsdomänen, bei denen Stigmergie vorzufinden ist. Hier können Prozesse, welche die gesamte Umgebung einer Anwendung betreffen und für das Koordinationsverhalten von Softwarekomponenten von Interesse sind, in Koordinationsmedien gekapselt und verarbeitet werden. Konkret könnte man dies z.B. für digitale Pheromone nutzen. Hier würde das Koordinationsmedien die Prozesse der Propagation und der Evaporation verarbeiten. Des Weiteren sind Anwendungen denkbar, bei denen räumliche Nachbarschaftsbeziehungen zwischen den Agenten bei der Koordination von Bedeutung sind. Auch hier könnte diese Funktionalität über ein Koordinationsmedium erbracht werden. In diesem Sinne können solche aktiven Koordinationsmedien auf Grund ihrer Funktionalität auch als programmierbare Koordinationsmedien bezeichnet werden (vgl. [Vil13]).

Nachdem das Koordinationsereignis vom Koordinationsmedium im *Coordination Space* publiziert wurde, generiert dieser einen sogenannten *Space Event*, über den das Koordinationsereignis von den *Percept*-Generatoren wahrgenommen werden kann. Diese generieren nun entsprechende *perceptions* für die vom *Coordination Space* unterstützten BDI- und Task-Agenten. Die *perceptions* enthalten dabei das Koordinationsereignis. Ein solches auf die Architektur des zu koordinierenden Agenten angepasstes Koordinationsereignis kann dann vom entsprechenden *Percept*-Prozessor im Agenten weiter verarbeitet werden.

Eine ausführlichere Beschreibung der Konzepte, wie Agenten Änderungen in der Umgebung im Rahmen des dem *Coordination Space* zugrundeliegenden *Environment Support* wahrnehmen und verarbeiten können, findet sich in [JBP10].

### Trennung von Aspekten

Mit Blick auf die Beziehung zwischen Kern- und Koordinationsfunktionalität verfolgt der *Coordination Space* Ansatz eine Trennung beider Aspekte. Die jeweiligen Funktionalitäten können dabei in einem hohen Maß unabhängig voneinander entwickelt werden, da sie nur lose gekoppelt sind. Der Aspekt der Koordination wird betont und ist ein Element erster Ordnung bei der Modellierung und Implementierung. Im Bezug auf die in Abschnitt 3.2.2 diskutierten Entwicklungsstrategien gehört die Integrationsarchitektur des *Coordination Space* zu der Klasse der Ansätze, die Koordination auslagern. Dadurch ist dieser Ansatz besonders für dezentrale selbstorganisierende Anwendungen von Interesse, bei denen die Koordinationsfunktionalität ein Kernaspekt ist. Bei Anwendungen dieser Klassen werden die Koordinationsmedien oft angepasst und ausgetauscht. Der *Coordination Space* unterstützt sowohl dies als auch die Implementierung und Wartung der Kernfunktionalität der Softwarekomponenten. Letztere müssen in dem Ansatz nicht ständig angepasst werden, wenn die Koordinationsfunktionalität angepasst wird. Dieses wird in

der Integrationsarchitektur dadurch deutlich, dass bis auf die Koordinationsmedien alle dortigen Komponenten so aufgebaut sind, dass sie unabhängig von der tatsächlichen Anwendungsdomäne und der Art und Weise der Koordination sind.

In den Koordinationsmedien wird daher die jeweils spezifische Selbstorganisationsfunktionalität realisiert, die den dynamischen und austauschbaren Teil der Integrationsarchitektur bildet. Koordinationsmedien können in diesem Kontext auch unabhängig von den Agenten eigene Berechnungen durchführen und sind nicht nur passive Entitäten, die lediglich zur Kommunikation zwischen Agenten genutzt werden.

### Verteilung

Die Grundform der Integrationsarchitektur des *Coordination Space* unterstützt dezentral selbstorganisierende Anwendungen, ist dabei aber konzeptionell auf die Ausführung auf einen Netzwerkknoten limitiert. Diese Einschränkung wurde im Rahmen dieser Dissertation aufgehoben, indem die Integrationsarchitektur so weiterentwickelt wurde, dass auch eine verteilte Ausführung möglich ist [PVR13].

Abbildung 4.2 verdeutlicht auf konzeptioneller Ebene diese Erweiterung und zeigt die Nutzung des *Coordination Space* in dem Kontext einer verteilten Anwendung. Diese besteht aus einer Menge von Softwarekomponenten, in diesem Fall Agenten, die sich auf unterschiedlichen Netzwerkknoten befinden. Die Koordinationsfunktionalität der verteilten Anwendung ist dabei jeweils in lokalen Koordinationsräumen ausgelagert, die der Grundform der Integrationsarchitektur entsprechen. Der *Coordination Endpoint* (CE in der Abbildung) kapselt dabei diejenigen Teilkomponenten des *Coordination Space*, die zur Observierung und Benachrichtigung einer Softwarekomponente benötigt werden. Das Koordinationsmedium kapselt die lokale Verarbeitung der Koordinationsfunktionalität.

Um nun auch die Koordinationsfunktionalität einer Anwendung verteilen zu können, können die Koordinationsmedien durch einen *Medium Endpoint* erweitert werden. Diese Komponente hat die Aufgabe, die in einem Koordinationsmedium lokal eintreffenden *Coordination Events* an andere Netzwerkknoten zu propagieren, auf denen die verteilte Anwendung ebenfalls läuft. Gleichzeitig nimmt ein *Medium Endpoint* auch extern, d.h. von anderen Netzwerkknoten eintreffende *Coordination Events* an und leitet sie zur Verarbeitung an das lokale Koordinationsmedium weiter. Somit agieren die *Medium Endpoints* als Adapter für lokale Koordinationsmedien. Zudem bilden sie untereinander ein Kommunikationsnetzwerk, dessen Topologie von der konkreten Implementierung der *Medium Endpoints* abhängt. Grundsätzlich sind hier sowohl zentrale als auch dezentrale Lösungen möglich. Letztere erscheinen allerdings besser geeignet, um dynamische und selbstorganisierende Anwendungen umzusetzen.

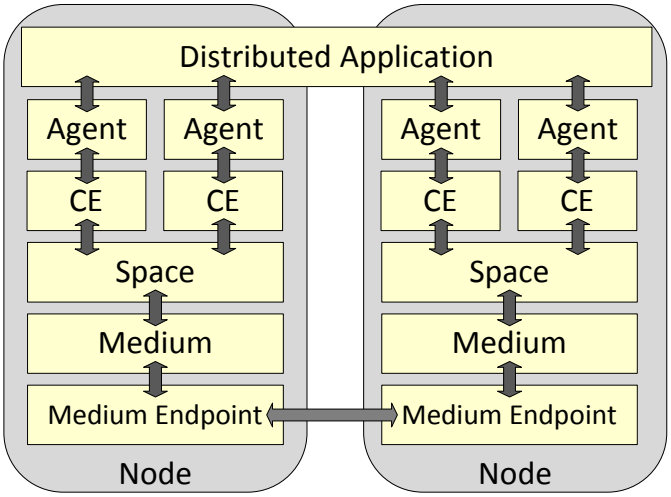



**Abbildung 4.2.:** Konzeptionelle Erweiterung der Integrationsarchitektur zu einem verteilten Coordinaton Space - aus (PVR13).

Bei dezentralen Netzwerk-Topologien sind Peer-to-Peer Ansätze attraktiv, da dort alle Netzwerkknoten gleichberechtigt agieren. Sie sind gleichzeitig Client und Server, d.h. die Interaktion erfolgt symmetrisch. Peer-to-Peer-Netzwerkarchitekturen lassen sich des Weiteren in strukturierte und unstrukturierte Topologien unterscheiden (vgl. [TvS07]). Bei strukturierten Topologien kann durch den Einsatz von sogenannten Overlay-Netzwerken eine übergeordnete logische Struktur, beispielsweise ein Ring, Stern, Baum etc., erzeugt werden. Im Kontext von verteilten *Coordination Spaces* können Overlay-Netzwerke von Interesse sein, wenn dadurch beispielsweise die Effizienz der Kommunikation zwischen den *Medium Endpoints* gesteigert werden kann. Es hängt somit von der konkreten Zielsetzung des Koordinationsmediums und der damit verfolgten Koordinationsfunktionalität ab, welche Netzwerktopologie die *Medium Endpoints* bilden.

Diese Erweiterung zur Verteilung der Integrationsarchitektur des *Coordination Space* ist transparent bezüglich der Koordinationsfunktionalität. Die Kernkomponenten des *Coordination Space* als auch die an der Koordination teilnehmenden Softwarekomponenten haben keine Information darüber, ob die Koordination nur auf einem Netzwerkknoten lokal erfolgt oder ob mehrere Netzwerkknoten beteiligt sind.

### 4.3.2. Ausführungsumgebung

Basierend auf dem im vorherigen Abschnitt vorgestellten konzeptuellen Modell des *Coordination Space* wird im Folgenden eine konkrete Realisierung präsentiert. Damit soll exemplarisch gezeigt werden, wie auf Basis der abstrakten Integrationsarchitektur die Implementation dezentral selbstorganisierender Systeme erfolgen kann. Die realisierte Ausführungsumgebung des *Coordination Space* [VSL+10] ermöglicht dabei die Implementierung selbst-

organisierender Systeme für beliebige Anwendungsdomänen und unterstützt die Nutzung unterschiedlicher Koordinationsmechanismen. Eine wichtige Erweiterung der Ausführungsumgebung wurde im Rahmen dieser Dissertation (siehe [PVR13]) umgesetzt und erlaubt eine Verteilung der Koordinationsräume über unterschiedliche Agentenplattformen und Netzwerkknoten hinweg.

### Programmierplattform

Um die Ausführungsumgebung des *Coordination Space* zu realisieren, wird eine Programmierplattform benötigt, welche die Implementation der ausgelagerten Koordinationsfunktionalität unterstützt. Prinzipiell kann diese Anforderung von den meisten bestehenden Programmiersprachen geleistet werden. In diesem Kontext haben *Vilenica et al.* [VSL+10] das bereits in Abschnitt 3.1.3 beschriebene Agentenrahmenwerk *Jadex* [PBJ13] als geeignet identifiziert, um Softwarekomponenten zu implementieren, mit denen sich dezentral selbstorganisierende Anwendungen realisieren lassen. Ein wesentliches Alleinstellungsmerkmal von *Jadex* ist dabei die Unterstützung unterschiedlicher Softwarekomponenten-Architekturen. So lassen sich mit Agenten, Services und Prozessen wichtige Konzepte verteilter Systeme nutzen und umsetzen.

Durch diese Flexibilität im Hinblick auf die Bereitstellung der Ausführungsumgebung des *Coordination Space* können mit einer einzigen Implementation potenziell viele Anwendungsdomänen unterstützt werden, sodass nicht jeweils separate Ausführungsumgebungen entwickelt werden müssen. Dadurch, dass *Jadex* auf der weitverbreiteten betriebssystemunabhängigen Programmiersprache *Java*[1] basiert, können einmal implementierte Anwendungen auf potenziell vielen Hard- und Softwareplattformen eingesetzt werden. Ein weiterer Vorteil ergibt sich aus der reichhaltigen Werkzeugunterstützung der *Jadex*-Plattform, welche die Entwicklung der Koordinationsfunktionalität erleichtert.

### Applikationskonzept

Eines der Kernkonzepte von *Jadex* ist die Möglichkeiten der Definition von sogenannten *Applikationen* (engl. *Applications*). Mit diesem Ansatz verfolgt *Jadex* insbesondere zwei Ziele (vgl. [PB10]): Zunächst soll es einfach möglich sein, zu spezifizieren, aus welchen Softwarekomponenten sich eine Anwendung zusammensetzt und diese ohne viel Aufwand um weitere Komponenten zu erweitern. Weiterhin soll es mit dem Konzept möglich sein, von der konkreten internen Architektur und Umsetzung der Softwarekomponenten zu abstrahieren. Eine Applikation ist dadurch ein Element erster Ordnung bei der Modellierung und Implementation verteilter Systeme und somit nicht nur das Ergebnis, das sich ergibt, wenn man alle einzelnen Softwarekomponenten zusammengefasst betrachtet.

[1] `https://www.java.com`, aufgerufen am 1. Februar 2017

Dieses Konzept wurde im Rahmen des *Coordination Spaces* aufgegriffen, indem eine Applikation als oberster logischer Container angesehen wird, den es bei der Modellierung und Implementierung dezentral selbstorganisierender Systeme zu betrachten gilt. Eine Applikationsspezifizierung definiert dabei hauptsächlich die Informationen über den Typ einer Anwendung, d.h. insbesondere die zugrundeliegende Struktur der Laufzeitkomponenten. Im Kontext der *Jadex*-Plattform gehören neben den eigentlichen Softwarekomponenten, wie z.B. Agenten und Services, also Komponenten, die hauptsächlich die Kernfunktionalität einer Applikation abbilden, auch Komponenten zur Visualisierung der Ausführungs- und Zeitsteuerung dazu. Die genaue Struktur einer Applikationsspezifizierung wird dabei deklarativ durch ein XML-Schema beschrieben.

### Umgebungen

Neben dem wichtigen Applikationskonzept bietet *Jadex* mit dem *Enviroment Support* [JBP10] ein weiteres Konzept an, das bei der Umsetzung der Ausführungsumgebung des *Coordination Space* von Interesse ist. Der *Environment Support* ermöglicht u.a. die explizite Nutzung von Umgebungen und leitet dabei ihre Modellierung, Strukturierung und Implementierung an (vgl. [Vil13]). Ferner ist die Anzahl von Umgebungen in einer Applikation nicht limitiert. Dadurch können jeweils eigenständige Umgebungen für spezifische Aspekte einer Anwendung eingesetzt werden. Abbildung 4.3 verdeutlicht diese Ausprägung exemplarisch. Dabei wird gezeigt, dass eine *Jadex*-Anwendung drei Umgebungen besitzt. Der *GridWorld-Space* ist zuständig für räumliche Aspekte innerhalb der Anwendung, während sich der *Organisational-Space* um organisatorische Beziehungen kümmert. Der *Coordination Space* schließlich zielt auf die Bereitstellung von Koordinationsfunktionalität in der Anwendung ab. Neben diesen drei Typen von Umgebungen, die alle Teil der Anwendung sind, wird in Abbildung 4.3 auch deutlich, dass die Softwarekomponenten, in diesem Fall Agenten, konzeptionell so aufgebaut sind, dass sie mit mehreren Umgebungen interagieren können. Hierfür wird vom *Environment Support* das Konzept von *Avataren*, die als Stellvertreter für die Softwarekomponenten in der Umgebungen agieren, verwendet.

### Realisierung

In diesem Abschnitt wird überblicksartig auf die konkrete Realisierung des Ausführungsumgebung des *Coordination Space* eingegangen werden. Dabei werden die wesentlichen Komponenten anhand des exemplarischen Ablaufs eines Koordinationsvorgangs beschrieben werden.

Ausgangspunkt der Realisierung der Ausführungsumgebung des *Coordination Space* ist die Java-Klasse `CoordinationSpace`. Diese Klasse wird von der Jadex-Laufzeitumgebung instanziiert, sobald eine Anwendung gestartet wird, die diese Umgebung in ihrer Spezifikation deklariert hat. Die Klasse durchläuft dabei nach dem Start zunächst eine Initialisierungsmethode, die

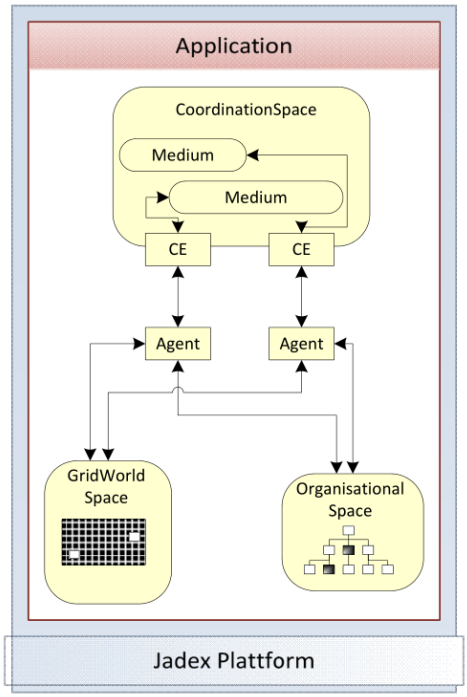



**Abbildung 4.3.:** Der Coordination Space und weitere Umgebungen als Teil einer Jadex-Anwendung - aus (Vil13, S. 98).

dafür zuständig ist, die in dieser Klasse gekapselte Koordinationsfunktionalität für die Anwendung bereitzustellen. Hierzu zählen insbesondere drei Aufgaben:

1. Eine das Interface `ICoordinationSpaceService` implementierende Klasse wird gestartet. Diese ist für das Management der Koordinationsmedien zuständig, die in einem *Coordination Space* genutzt werden. Dazu zählen insbesondere das (De-)Aktivieren von Koordinationsmedien sowie ihre Rekonfiguration (Aspekte, die in Abschnitt 4.4 im Rahmen der strukturellen Adaption von Koordinationsprozessen beschrieben werden).

2. Der *Coordination Space* wird so vorbereitet, wie es die Spezifikation für Umgebungen in *Jadex* vorsieht. Dieses bezieht sich insbesondere auf die Softwarekomponenten, die dem *Coordination Space* zur Laufzeit hinzugefügt werden. Hier muss sichergestellt werden, dass die Komponenten entsprechend initialisiert werden, damit sie an der Koordination teilnehmen können. Dieses geschieht transparent und nicht invasiv.

3. Die eigentliche Funktionsfunktionalität wird initialisiert. Hierzu wird das zugrundeliegende Koordinationsmodell, das in der Koordinationssprache *MASDynamics* [SR09b] definiert ist, von der Klasse `InitDeco4MAS` eingelesen und in eine entsprechende Java-Repräsentation überführt. Dadurch weiß der *Coordination Space*, welche

Typen von Softwarekomponenten an welcher dezentralen Koordination teilnehmen. Dabei werden außerdem die notwendigen *Percept*-Typen und -Generatoren vorbereitet.

Nachdem die Initialisierungsphase abgeschlossen ist, ist der *Coordination Space* bereit, die Koordinationsfunktionalität innerhalb der Anwendung zu erbringen. Immer, wenn eine Softwarekomponente der betrachteten Anwendung von der *Jadex*-Ausführungsumgebung gestartet wird, überprüft der *Coordination Space*, ob diese gemäß Koordinationsmodell am Koordinationsprozess beteiligt ist. Wenn dies der Fall ist, wird eine spezielle Klasse aufgerufen, welche die Observierung dieser Softwarekomponenten vorbereitet. Die Klasse hängt dabei von der zugrundeliegenden Architektur der Komponente ab. Handelt es sich z.B. um einen BDI-Agenten, wird die Java-Klasse `InitBDIAgentForCoordination` aufgerufen, während die Java-Klasse `InitMicroAgentForCoordination` für einfache Mikro-Agenten (Task-orientiert) zuständig ist. Die beiden Klassen kennen jeweils die speziellen Charakteristika der Architekturen und analysieren zur Vorbereitung der Observierung die Java-Repräsentation des zugrundeliegenden Koordinationsmodells. Dort ist jeweils definiert, welche(s) Element(e) einer Softwarekomponente observiert werden soll(en).

Für jedes definierte Element wird zur Observierung ein Listener instanziiert. Für den Fall eines BDI-Agenten erfolgt dies durch die Java-Klasse `BDIBehaviorObservationComponent`. Diese ist in der Lage, folgende spezifische Elementtypen zu observieren: *Belief*, *Belief-Set*, *Goal*, *Plan* und *InternalEvent*. Gilt es, einen Mikro-Agenten zu observieren, so ist hierfür die Java-Klasse `MicroBehaviourObservationComponent` zuständig. Das Verhalten eines Mikro-Agenten setzt sich aus einer Reihe von sogenannten *Component Steps* zusammen. Programmiertechnisch betrachtet sind das Klassen, die das `IComponentStep` Interface implementieren. Da Mikro-Agenten eine sehr schlichte Architektur haben, können hier lediglich spezielle `CoordinationComponentSteps` observiert werden. Die Ausführung dieser Komponentenschritte werden von den Scheduling-Mechanismen der *Jadex*-Plattform entsprechend verwaltet und geplant. Bei den `CoordinationComponentSteps` handelt es sich um eine spezielle Klasse von Komponentenschritte, die das `ICoordinationComponentStep` Interface implementieren, das wiederum eine Erweiterung des `IComponentStep` Interfaces darstellt. Die Erweiterung beschränkt sich dabei auf einer Markierungsfunktionalität, sodass bei der Modellierung und bei Observierung zur Laufzeit zwischen normalen Komponentenschritten und solchen, die für die Koordination relevant sind, unterschieden werden kann.

Nachdem die Observierung der an der Koordination teilnehmenden Softwarekomponenten angelaufen ist, wird ein *Coordination Event* von den jeweiligen Observern generiert, sobald ein im Koordinationsmodell definiertes Ereignis bzw. Zustand eingetroffen ist. Wird nun, nachdem die Vorbedingungen erfüllt sind, ein *Coordination Event* vom Observer generiert, so wird er an den

umgebenden *Coordination Space* weitergeleitet. Dies geschieht durch die *Publish*-Schnittstelle des Koordinationsraums. Im *Coordination Space* wird dann auf Grund der Verwaltungsinformationen des jeweiligen *Coordination Events* entschieden, an welches Koordinationsmedium dieser weitergereicht wird. Die Koordinationsmedien wiederum besitzen dafür eine *Perceive*-Schnittstelle, über die sie die Koordinationsereignisse wahrnehmen können.

Ab diesem Punkt ist der Verlauf des Koordinationsprozesses lediglich von der konkreten Semantik des Koordinationsmediums abhängig. Im Fall von programmierbaren Koordinationsmedien können hier auch komplexe Berechnungen erfolgen. Der Koordinationsprozess wird dann fortgesetzt, wenn das Koordinationsmedium über die *Publish*-Schnittstelle des umgebenden *Coordination Space* ein *Coordination Event* sendet, der als Ergebnis der Berechnungen angesehen werden kann. Das Verhältnis zwischen der Anzahl der in ein Koordinationsmedium eingehenden und wieder ausgehenden *Coordination Events* ist dabei lediglich von der Semantik des Koordinationsmediums abhängig. So können auf einen eingehenden *Coordination Event* keiner oder beliebig viele ausgehende folgen. Selbiges gilt auch für die zeitliche Beziehung zwischen eingehenden und ausgehenden *Coordination Events*. Auch diese Beziehung lässt sich nicht verallgemeinern, sondern ist spezifisch für jedes Koordinationsmedium.

Nachdem ein *Coordination Event* dem umgebenden C*oordination Space* vom Koordinationsmedium mitgeteilt wird, generiert der *Coordination Space* daraus einen `EnvironmentEvent`. Dieses ist ein allgemeines Element in *Jadex*, mit dem Umgebungen ereignisorientiert mit Softwarekomponenten kommunizieren können. Damit erfolgt die Kommunikation asynchron und ist lose gekoppelt. Der erzeugte `EnvironmentEvent` kann wiederum von speziellen *Jadex*-Komponenten weiterverarbeitet werden. Die Komponenten agieren dabei immer paarweise. Ein allgemeiner `EnvironmentEvent` wird von einer Klasse, die das Interface `IPerceptGenerator` implementiert, in eine spezielle Wahrnehmung transformiert. Diese wiederum kann von einer Klasse verarbeitet werden, die das `IPerceptProcessor` Interface implementiert. Im Fall der Ausführungsumgebung des *Coordination Space* wird dieses Paar durch die Java-Klassen `DefaultCoordinationEventGenerator` und `DefaultCoordinationInformationInterpreter` realisiert. Diese werden bereits in der Initialisierungsphase in der Klasse `InitDeco4MAS` gestartet und agieren dabei als Listener für die Erzeugung von Umweltereignissen. Der `DefaultCoordinationEventGenerator` ist dabei insbesondere dafür zuständig, dass von den generierten Umweltereignissen in einer Umgebung nur diejenigen weiterverarbeitet werden, die für die Koordinationsfunktionalität benötigt werden. Diese werden dann auf Basis der enthaltenen Verwaltungsinformationen aufbereitet und es wird eine entsprechende Wahrnehmung generiert, die vom `DefaultCoordinationInformationInterpreter` weiterverarbeitet wird. Der `DefaultCoordinationInformationInterpreter` arbeitet dabei direkt auf den an der Koordination beteiligten Softwarekomponenten und ist in

der Lage, den internen Zustand der Komponenten zu ändern. Dazu werden Informationen der erhaltenen Wahrnehmung herangezogen. Der Vorgang ist dabei für die Softwarekomponenten transparent und erfolgt nebenläufig, die Kernfunktionalität ist davon also zunächst nicht beeinträchtigt. Gleichwohl kann das Ergebnis der Änderung allerdings das Verhalten der Softwarekomponente beeinflussen. Beispielsweise kann die Verarbeitung einer solchen Wahrnehmung in einem BDI-Agenten dazu führen, dass der Wert eines *Beliefs* verändert wird. Dadurch kann gleichzeitig ein interner Trigger im Agenten aktiv werden, der wiederum zur Konsequenz hat, dass ein neues *Goal* gestartet wird. Während der Agent dieses Ziel verfolgt, kann dann ein Zustand eintreten, der den Observer, in diesem Fall die `BDIBehaviorObservationComponent`, aktiviert. Damit beginnt dann erneut ein Koordinationsablauf, wie er hier exemplarisch dargestellt wurde.

Die exemplarische Umsetzung der Integrationsarchitektur des *Coordination Space* auf Basis der *Jadex*-Plattform zeigt, wie die Implementation dezentral selbstorganisierender Anwendungen systematisch unterstützt werden kann (vgl. [VSL+10]). Der vorgestellte Ansatz unterstützt die Trennung von Kernfunktionalität und Koordinationsfunktionalität bei der Programmierung. Gleichzeitig werden zur Implementierung von Koordinationsfunktionalität keine neue Technologien benötigt, da die Ausführungsumgebung des *Coordination Space* ein integraler Bestandteil der eigentlichen Programmierplattform ist. Dies bedeutet, dass dezentrale Koordinationsfunktionalität als ein Design- und Implementationsaspekt erster Ordnung betrachtet wird und hierbei auf Programmierebene dieselbe Unterstützung bei der Entwicklung erfährt wie gewöhnlicher *Jadex*- bzw. Java-Code, der Kernfunktionalität beinhaltet. Dadurch muss für die Implementation dezentraler Koordinationsfunktionalität kein neues bzw. spezifisches Entwicklungswerkzeug und keine neuartige Entwicklungssprache verwendet werden. Stattdessen können etablierte Werkzeuge und auch die weit verbreitete und betriebssystemunabhängige Programmiersprache Java verwendet werden. Insgesamt sinkt dadurch der Aufwand für die Einarbeitung, Umsetzung und Wartung dezentral selbstorganisierender Anwendungen erheblich, da alle wesentlichen Aspekte auf Basis einer etablierten Plattform umgesetzt werden können. Zugleich zeichnet sich der Ansatz aber auch durch eine hohe Flexibilität und Wiederverwendbarkeit aus (vgl. [VSL+10]).

### Verteilung

Neben der grundsätzlichen Form der Ausführungsumgebung des *Coordination Space*, die im Vorangegangenen vorgestellt wurde, wurde im Rahmen dieser Dissertation eine Erweiterung zur Verteilung des Konzepts für mehrere Netzwerkknoten hinweg entwickelt (siehe Abschnitt 4.3.1). Diese wird für Anwendungsszenarien benötigt, die aufgrund von örtlicher Verteilung nicht auf einer lokalen Plattform ausgeführt werden. Weiterhin ist die Erweiterung für Si-

tuationen von Interesse, bei denen Anwendungen aufgrund ihrer Größe und Komplexität auf mehreren Rechnern verteilt ausgeführt werden sollen.

Wie bereits bei der Konzeption der Integrationsarchitektur des *Coordination Space* im vorangegangenen Unterabschnitt diskutiert wurde, wird die Grundform erweitert, um so die Verteilung zu ermöglichen. Dabei wurde auf konzeptioneller Ebene die Komponente des *Medium Endpoint* eingeführt, die maßgeblich zur Verteilung von Koordinationsräumen beiträgt. Abbildung 4.4 zeigt die Übertragung dieses Konzepts für die Ausführungsumgebung des *Coordination Space* auf Basis der *Jadex*-Plattform. Hierzu sind insbesondere zwei Anpassungen notwendig: Die Erste ist erforderlich, da das Anwendungskonzept der zugrundeliegende *Jadex*-Plattform von sich aus keine Unterstützung für verteilte Anwendungen bietet. Daher wird das Element des *Coordination Context* eingeführt. Jede auf einem Netzwerkknoten laufende *Jadex*-Anwendung kann in ihrer Spezifikation eine entsprechende `CoordinationContextID` definieren, um damit zu signalisieren, zu welchem Koordinationskontext sie gehört. Über den Koordinationskontext werden also die lokale *Jadex*-Anwendungen, die jeweils auf einer Plattform ausgeführt werden, zu einem gemeinsamen, verteilten Koordinationskontext zusammengefasst. Somit bilden alle *Jadex*-Anwendungen mit derselben `CoordinationContextID` eine verteilte Anwendung im Hinblick auf die Koordinationsfunktionalität. *Jadex*-Anwendungen, die auf einem gemeinsamen Netzwerkknoten ausgeführt werden, können dabei entweder zur selben verteilten Anwendung gehören oder auch unterschiedlichen verteilten Kontexten zugeordnet sein.

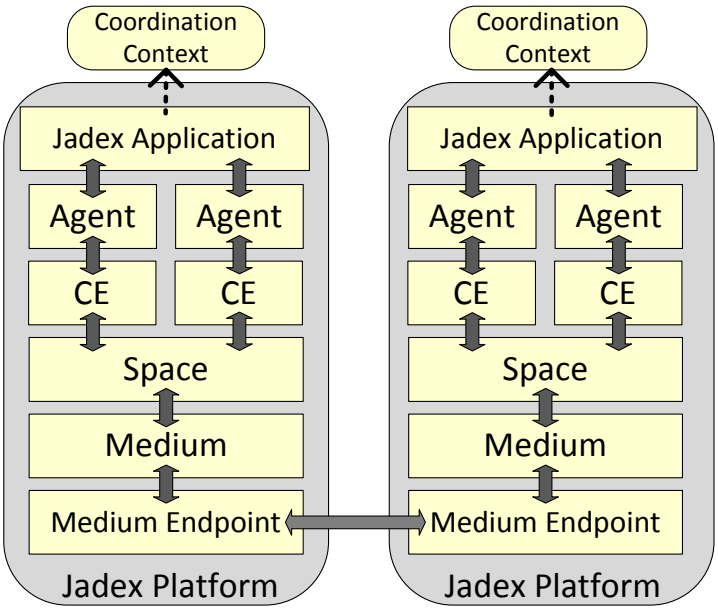



**Abbildung 4.4.:** Die Realisierung der verteilten Ausführungsumgebung des Coordination Space in Jadex - aus (PVR13, S. 8).

Die zweite Anpassung zielt auf die Fragegenstellung ab, wie sich nun die Koordinationsfunktionalität verteilen lässt. Hierzu bleibt im Prinzip die Grundfunktionalität der Ausführungsumgebung des *Coordination Space* erhalten. Somit ändert sich auch für eine verteilte *Jadex*-Anwendung die lokale Verarbeitung von Koordinationsfunktionalität nicht. Neu hinzu kommt, dass *Coordination Events*, die von den Observern der Softwarekomponenten generiert und über seine *Perceive*-Schnittstelle an den *Coordination Space* weitergeleitet werden, nun verteilt werden. Dabei leitet der *Coordination Space* wie bisher die *Coordination Events* anhand ihrer Verwaltungsinformationen an das entsprechende Koordinationsmedium weiter. Dieses ist nun neben der lokalen Verarbeitung der *Coordination Events* auch für deren Verteilung zuständig. Dazu bilden die Koordinationsmedien durch die Mediumendpunkte (*Medium Endpoint*) mit anderen Koordinationsmedien, die zu selben Jadex-Anwendung gehören, ein *Overlay*-Netzwerk. Die Topologie dieses Overlay-Netzwerks bestimmt damit auch die Charakteristik, mit der *Coordination Events* verteilt werden. Bei der Verteilung der *Coordination Events* zwischen Koordinationsmedien auf unterschiedlichen Netzwerkknoten dient die oben eingeführte `CoordinationContextID` dazu, die Mediumendpunkte zu identifizieren, die zu der gleichen verteilten *Jadex*-Anwendung gehören. Wenn ein Koordinationsmedium einen *Coordination Event* über das Overlay-Netzwerk empfängt, wird dieser genauso wie die lokalen *Coordination Events* verarbeitet und die Koordinationsfunktionalität erbracht.

Diese vorgestellte Erweiterung der Ausführungsumgebung des *Coordination Space* zur Unterstützung verteilter Jadex-Anwendungen hat den Vorteil, dass sie minimal invasiv ist. Wie in der Grundform werden Kernfunktionalität und Koordinationsfunktionalität getrennt. Auch erfordert die Verteilung einer Anwendung lediglich die Spezifikation der speziellen `CoordinationContextID`. Zudem ist die Verteilung für den eigentlichen *Coordination Space* transparent, da er von der Kommunikation der Mediumendpunkte nicht beeinflusst wird und keine Kenntnis über diese erlangt. Die Koordinationsmedien kapseln die Verteilung dabei vollständig. Zugleich lassen die Koordinationsmedien dem Entwickler einen sehr hohen Freiheitsraum für die konkrete Umsetzung der Verteilung. Deshalb sind neben Peer-To-Peer-Ansätzen, die besonders für dezentral selbstorganisierende Anwendungen von Interesse sind, prinzipiell auch zentrale Client-Server-Ansätze möglich.

### 4.3.3. Koordinationsmedien

In den vorangegangenen Abschnitt wurden die grundlegenden Aspekte der Ausführungsumgebung des *Coordination Space* vorgestellt. Dabei wurden insbesondere die Softwarekomponenten und ihre Implementierungen detailliert vorgestellt, die sowohl weitestgehend unabhängig von der Semantik der Anwendung als auch von der tatsächlichen Koordinationsfunktionalität sind. In Bezug auf die Umsetzung von Koordinationsfunktionalität ist dabei klar geworden, dass die spezifischen Charakteristika in den Koordi-

nationsmedien gekapselt sind. Dagegen sind die anderen Komponenten der Ausführungsumgebung des *Coordination Space* so konzipiert, dass sie die allgemeine Koordinationsfunktionalität vorhalten und somit nicht für neue Kontexte bzw. Anwendungen angepasst werden müssen (vgl. [VSL+10]). Im Folgenden werden beispielhaft konkrete Implementierungen von Koordinationsmedien vorgestellt, um einen Eindruck zu vermitteln, wie spezifische Koordinationsfunktionalität gekapselt werden kann. Der Fokus liegt dabei auf verteilten Ausprägungen, die im Rahmen dieser Dissertation entstanden sind.

### Remote Service Call

Das *Remote Service Call* Koordinationsmedium basiert auf der Idee, die bestehende Service-Infrastruktur der *Jadex*-Plattform zur Verteilung zu nutzen (vgl. [PVR13]). Hierbei wird ein *Medium Endpoint* realisiert, der die `ICoordinationService` Schnittstelle implementiert. Dieser Service wird im Sinne eines Remote Service Calls aufgerufen, wenn ein Coordination Event aus einem lokalen Coordination Space verteilt werden soll. Der daraus resultierende Ablauf ist in Abbildung 4.5 dargestellt.

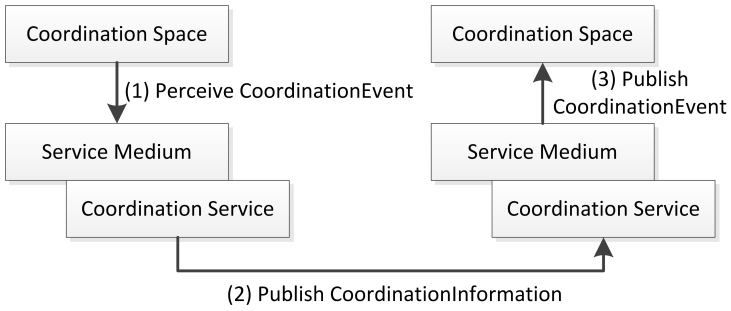



**Abbildung 4.5.:** Ein verteiltes Koordinationmedium basiered auf einem *Remote Service Call* Ansatz - aus (PVR13, S. 9).

Sobald ein Koordinationsmedium dieses Typs einen lokalen *Coordination Event* empfängt, wird dieser zunächst lokal verarbeitet (1). Anschließend verwendet das Koordinationsmedium seinen *Medium Endpoint*, um den Event zu verteilen. Dazu wird die *Jadex* Service Infrastruktur verwendet, die es erlaubt, im Netzwerk nach anderen *Medium Endpoints* zu suchen, die ebenfalls die `ICoordinationService` Schnittstelle implementieren. Dort wird dann die entfernte Methode zum Empfang von *Coordination Events* aufgerufen, sofern die Endpunkte zur selben verteilten *Jadex*-Anwendung gehören (2). Zur Identifikation wird die bereits erwähnte `CoordinationContextID` verwendet. Der empfangene *Coordination Event* wird schließlich auf dem entfernten Netzwerkknoten durch das verteilte Koordinationsmedium weiterverarbeitet (3). Aus konzeptioneller Sichtweise heraus gehört der Typ des Koordinationsmediums Remote Service Call zur Klasse der *Broadcast-Push*-Kommunikationsprotokolle.

### Publish/Subscribe

Das *Publish*/*Subscribe*-Koordinationsmedium [PVR13] realisiert aus konzeptioneller Sicht ein Kommunikationsprotokoll, das zur Klasse der *Multicast-Push*-Kommunikationsprotokolle gehört. Der *Medium Endpoint* eines solchen Koordinationsmediums implementiert die `ICoordinationSubscriptionService` Schnittstelle. Der schematische Ablauf der Koordinationshandlung für diesen Fall ist in Abbildung 4.6 dargestellt.

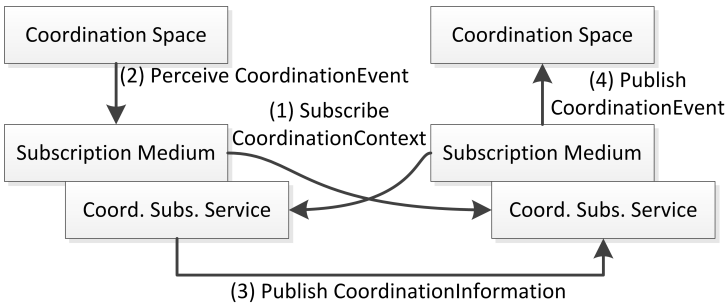



**Abbildung 4.6.:** Ein verteiltes Koordinationmedium basiered auf einem *Publish/Subscribe* Ansatz - aus (PVR13, S. 9).

Verglichen mit dem zuvor betrachteten Remote-Service-Call-Koordinationsmedium, besteht der Unterschied darin, dass der *Medium Endpoint* in diesem Fall selber aktiv nach anderen Instanzen der verteilten *Jadex*-Anwendung suchen muss, sofern er an der verteilten Koordination teilnehmen soll. Dazu sucht der Endpunkt während der Initialisierungsphase nach anderen Endpunkten, die ebenfalls die `ICoordinationSubscriptionService` Schnittstelle implementieren. Gehören diese zur gleichen `CoordinationContextID`, so kann sich der Endpunkt bei der entfernten Schnittstelle für den Empfang von *Coordination Events* dieser Plattform registrieren (1). Wenn lokal in einem *Coordination Space* ein *Coordination Event* erzeugt und von dem Publish/Subscribe Koordinationsmedium wahrgenommen wird (2), so wird dieser *Coordination Event* an alle Instanzen, die sich für den Empfang registriert haben, verteilt (3). Auf der entfernten Instanz wird der *Coordination Event* dann wieder lokal im *Coordination Space* mit Hilfe des dortigen *Medium Endpoints* weiterverarbeitet (4).

### Distributed Hash Table

Während die beiden vorangegangenen Typen die *Jadex*-Service-Infrastruktur zur Verteilung nutzen, verwendet dieser Ansatz mit *Scribe* [RKCD01] eine von der *Jadex*-Plattform unabhängige Technologie zur Verteilung der *Coordination Events* im Netzwerk [PVR13]. Bei *Scribe* handelt es sich um eine gut skalierende Event-basierte Infrastruktur, die Anwendungen eine Publish/Subscribe-Funktionalität zur Verfügung stellt. *Scribe* selber basiert

dabei auf *Pastry* [RD01], einem Ansatz, der es ermöglicht, ein Overlay-Netzwerk im Internet aufzubauen, um damit eine *Distributed Hash Table* [RR06] zu realisieren. Dazu bietet *Pastry* u.a. die Möglichkeit einer generischen Peer-to-Peer Lokation von Objekten an. Dieses wird durch Fähigkeiten zum Routing ergänzt. *Scribe* nutzt somit *Pastry*, um Gruppen für spezifische Themen zu erstellen. Außerdem wird damit ein effizienter Multicast-Baum zur Kommunikation erzeugt. Dieser wird dann dazu genutzt, um die Ereignisse einer Gruppe an die Softwarekomponenten zu senden, die sich für den Empfang registriert haben. Abbildung 4.7 zeigt die Nutzung von *Scribe* im Kontext der verteilten Ausführungsumgebung des *Coordination Space*.

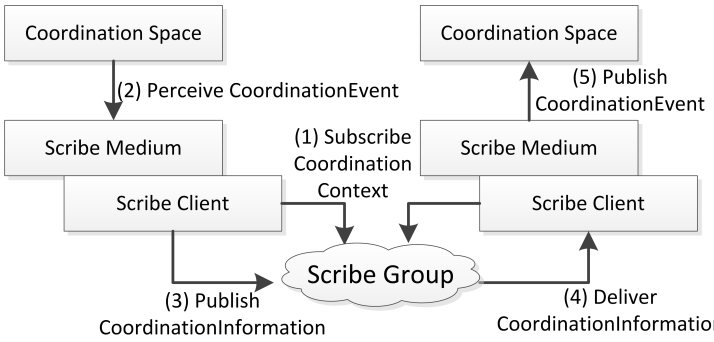



**Abbildung 4.7.:** Ein verteiltes Koordinationmedium basiered auf einem *Dishtributed Hash Table* Ansatz - aus (PVR13, S. 9).

Jedes verteilte Koordinationsmedium von diesem Typ besitzt einen *Scribe*-Client, der die Rolle des *Medium Endpoint* übernimmt. Mit Hilfe seiner `CoordinationContextID` registriert sich dieser in der entsprechenden *Scribe*-Gruppe für seinen Koordinationskontext (1). Bei dieser Implementation realisiert die Klasse `ScribeMedium` das entsprechende verteilte Koordinationsmedium. Diese Klasse empfängt vom umgebenden *Coordination Space* über ihre *Perceive*-Schnittstelle lokale *Coordination Events* (2). Diese werden an den *Scribe*-Client weitergeleitet, der sie in der entsprechenden *Scribe*-Gruppe publiziert (3). Die in der Gruppe registrierten *Scribe*-Clients erhalten die *Coordination Events* und publizieren diese dann wiederum weiter an ihr verteiltes Koordinationsmedium, wo sie schließlich weiterverarbeitet (4) und schließlich in den *Coordination Space* publiziert werden (5).

## 4.4. Strukturelle Adaption von Koordinationsprozessen

Das Konzept der Selbstorganisation ist seit mehr als einem Jahrzehnt als dezentraler Ansatz zur Entwicklung von selbst-adaptiven Systemen etabliert (vgl. u.a. [PR15]). Allerdings können selbstorganisierenden Systeme unter bestimmten Bedingungen ein unerwünschtes Verhalten, z.B. in Form eines Ab-

falls an Performanz oder der sogenannten *„Starvation“*[2], zeigen. Ein vielversprechendes Konzept zur Überwindung solcher dynamischen Ineffizienzen in selbstorganisierenden Systemen ist der dynamische Austausch bzw. die Rekonfiguration der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden. Ein solcher Austausch der Koordinationsprozesse oder deren Rekonfiguration wird im Kontext dieser Arbeit als *strukturelle Adaption* [PRD15] bezeichnet, da dadurch die Struktur des selbstorganisierenden Verhaltens angepasst wird.

In diesem Abschnitt werden eine Architektur und ein Entwicklungsansatz vorgestellt, die den strukturellen Austausch von selbstorganisierenden Koordinationsprozessen für selbstorganisierende Multiagentensysteme unterstützen. Konzeptionell wird dabei jeder Agent mit einem Sensor ausgestattet, der dedizierte Performanz-Indikatoren überwacht, die Aufschluss über die Performanz bzw. die Güte des selbstorganisierenden Verhaltens geben, und einen verteilen Adaptionsprozess auslöst, wenn sich die entsprechende Performanz verringert. Im Rahmen dieses Prozesses werden strukturelle oder parametrische Anpassungen an den Koordinationsprozessen vorgenommen, mit dem Ziel, dem unerwünschten selbstorganisierenden Verhalten entgegen zu steuern. Durch die Anpassungen an den Koordinationsprozessen soll es so auf einer übergeordneten Ebene, ermöglicht werden, die Struktur des Selbstorganisationsprozesses an sich anzupassen. Dies geschieht dadurch, dass Teilstrukturen aktiviert und deaktiviert werden. Ein Beispiel, wie einem ungewünschten selbstorganisierenden Verhalten durch ein künstlich erzeugtes Phänomen entgegengewirkt werden kann findet sich z.B. in [SR10a].

Wie bereits im Verlauf dieser Arbeit erörtert wurde, stellen die Entwicklung und der Betrieb von adaptiven Systemen eine große Herausforderung dar. Daher wird ein systematischer Entwicklungsansatz benötigt, der in allen Phasen in der Lage ist, die drei inhärenten Charakteristika dieser Systeme zu handhaben: Nichtlineare Dynamiken, stochastisches Verhalten und emergente Phänomene. Mit dem *SodekoVS*-Ansatz [SBP+09] (vgl. Abschnitt 3.2.3) wurde ein systematischer Ansatz zur Entwicklung selbstorganisierender Systeme entwickelt, der diese Herausforderungen adressiert. Mit dem Konstruktionsansatz des *Coordination Space* [VSL+10] wurde im vorherigen Abschnitt eine Realisierung der *SodekoVS*-Referenzarchitektur [SR09a] präsentiert, welche die Konzeption und Implementation dezentraler selbstorganisierender Systeme unterstützt. Grundlegend, sowohl für das *Coordination Space* Konzept als auch für den Ansatz der strukturellen Adaption von Koordinationsprozessen, ist das konzeptionelle Modell aus [Sud10], welches Adaptionsdynamik, Koordinationsprozesse und Koordinationsmechanismen in einer verteilten Anwendung in Beziehung zueinander setzt (vgl. Abschnitt 2.3.1). Gemäß diesem Modell lässt sich in einem verteilen System ein Verhalten beobachten, dass auf der Systemebene eine Dynamik erzeugt, die das Resultat eines adaptiven

[2] Englisch für *Verhungern*, bezeichnet in der Informatik einen Prozess, der warten muss, weil er auf eine Ressource wartet, auf die er aber nicht zugreifen kann, weil diese von einem anderen Prozess verwendet wird. Dieser Prozess kann daher nicht terminieren.

Verhaltens ist. Diese Adaptionsdynamik wird wiederum durch einen Koordinationsprozess beschrieben. Dieser Prozess ist dafür verantwortlich, dass sich das System kontinuierlich selbstorganisiert. Dabei beeinflusst bzw. strukturiert der Koordinationsprozess das Verhalten der einzelnen Systemkomponenten in dezentraler Weise. Die Fähigkeit, dieses zu tun, beruht dabei maßgeblich auf der Verwendung von dezentralen Koordinationsmechanismen. Diese Mechanismen erfüllen wiederum zwei Aufgaben. Sie stellen sowohl Techniken zum Informationsaustausch zwischen den Systemelementen zur Verfügung als auch auch Techniken welche die Anpassung der lokalen Systemelemente ermöglichen. Der Ansatz der strukturellen Adaption [PR15, PRD15] stellt nun den Austausch dieser Koordinationsprozesse bzw. deren Rekonfiguration in den Vordergrund. Dadurch wird ein verteiltes System in die Lage versetzt, nicht nur sein Verhalten, sondern auch seine inhärente Struktur anzupassen, um Performanzprobleme wie z.B. *„Starvation“* zu überwinden, indem die Struktur der selbstorganisierenden Prozesse automatisiert angepasst wird. Konzeptionell ist dieser Ansatz dabei vergleichbar mit dem lokalen, reaktiven Planen aus dem BDI-Agenten Software Modell [RG+95], bei dem ein Agent einen Plan basierend auf seinen lokalen Informationen und Alternativen auswählt. Der Ansatz der strukturellen Adaption beschäftigt sich dabei mit der Auswahl einer geeigneten, vordefinierten Adaption der Koordinationsprozesse, die auf anwendungsglobaler Ebene auf einem verteilten Konsens der beteiligten Komponenten basiert. Dabei trifft jede der beteiligten Komponenten ihre Entscheidung basierend auf ihren lokalen Informationen und es wird ein dezentraler Ansatz verwendet, um einen verteilten Konsens zu finden.

Betrachtet man adaptive Systeme im Detail, so offenbaren sie eine Reihe von unterschiedlichen Facetten. So wird z.B. in [Gei08] zwischen Adaptivität zur Entwurfs- und zur Laufzeit unterschieden, wobei letztere wesentliche anspruchsvoller in der Umsetzung ist. In Abbildung 4.8 werden weitere Dimensionen adaptiver Systeme aufgezeigt. Die Ansätze werden hierbei danach unterschieden, ob im Zuge der Adaption nur einzelne Systemparameter angepasst werden oder die gesamte Struktur adaptiert wird. Des Weiteren wird zwischen Ansätzen unterschieden bei denen das selbstorganisierende Verhalten durch zentrale oder dezentrale Steuerungskonzepte umgesetzt wird. Die dritte Dimension gibt schließlich Auskunft darüber, ob die Adaption automatisiert oder manuell ausgeführt wird. Der rote Punkt in Abbildung 4.8 ordnet nun den Ansatz zur *strukturellen Adaption* von Koordinationsprozessen innerhalb dieser Dimensionen adaptiver Systeme ein. Der Ansatz basiert auf einem dezentralen Steuerungskonzept, bei dem ohne zentrale Komponente über mögliche strukturelle Adaptionen entschieden wird. Die Adaptionen ermöglichen es dabei, Anpassungen an der Struktur des Systems vorzunehmen (wobei parameter-basierte Anpassungen ebenfalls möglich sind). Diese möglichen Anpassungen, werden manuell von einem Entwickler zur Design-Zeit modelliert und in Form von vordefinierten Adaptionsregeln und -anweisungen automatisiert zur Laufzeit ausgeführt.

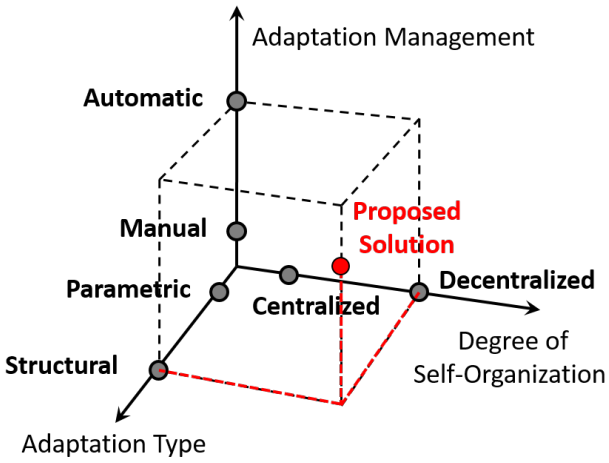



**Abbildung 4.8.:** Einordnung des Ansatzes zur strukturellen Adaption von Koordinationsprozessen (roter Punkt) in die unterschiedlichen Dimensionen adaptiver Systeme - aus (PRD15, S. 1).

### 4.4.1. Strukturelle Adaption in selbstorganisierenden Multiagentensysteme

Das im Vorangegangenen beschriebene Zusammenspiel selbstorganisierender Dynamiken und der strukturellen Adaption von Koordinationsprozessen in selbstorganisierenden Multiagentensysteme wird in Abbildung 4.9 nochmal verdeutlicht. Die linke Seite der Abbildung zeigt dabei, wie selbstorganisierende Dynamiken nach [Sud10] in einem Multiagentensystem realisiert werden können (vgl. dazu auch Abbildung 2.6). Aus konzeptioneller Sicht bedeutet eine strukturelle Adaption in einem selbstorganisierenden System eine Adaption der Koordinationsprozesse, da diese die vorgesehene selbstorganisierende Dynamik des Systems beschreiben. Illustriert wird dieses Konzept auf der rechten Seite von Abbildung 4.9. In diesem Fall weist das selbstorganisierende Multiagentensystem die Fähigkeit zur strukturellen Adaption auf. Vergleichbar mit der selbstorganisierenden Dynamik ist diese durch Prozesse beschrieben. Diese Prozesse definieren Adaptionsbedingungen, die wiederum einen spezifischen Zustand des Systems beschreiben, unter denen die selbstorganisierende Dynamik angepasst werden soll. Diese Anpassungen werden durch vorgeschriebene Adaptionsaktivitäten realisiert. Dabei werden z.B. ineffektive Koordinationsprozesse deaktiviert und andere Koordinationsprozesse dafür im Austausch aktiviert, um dem negativen Systemverhalten entgegen zu wirken. Das Ergebnis hiervon ist dann eine strukturelle Adaption der Koordinationsprozess-Komposition. Für den Fall, dass das Systemverhalten zwar ineffektiv, aber nicht direkt fehlerhaft wird, ist auf einem einfacheren Level auch eine Rekonfiguration der aktiven Koordinationsmedien in Bezug auf ihre Koordinationsparameter möglich. Dabei wird nicht direkt die Struktur der selbstorganisierenden Dynamik angepasst, sondern nur einzelne Parameter der Koordinationsprozesse verändert, um deren Verhalten effektiver zu gestalten.

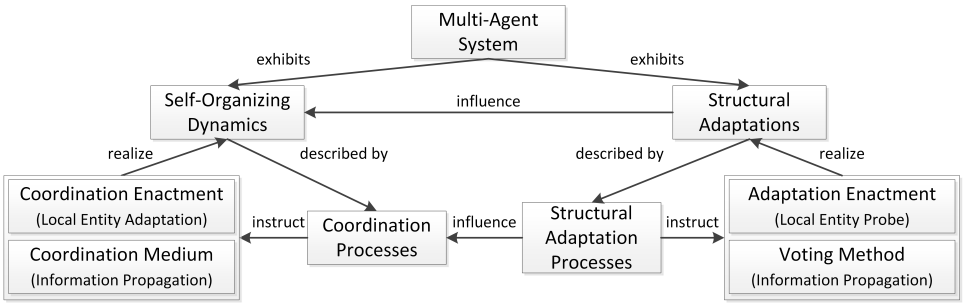



**Abbildung 4.9.:** Einordnung struktureller Adaptionsprozesse in dem Kontext eines selbstorganisierenden Multiagentensystems - nach (PRD15).

Ähnlich wie die Koordinationsprozesse, die *Coordination Enactments* anweisen, um im Zuge der selbstorganisierenden Dynamik lokale Änderungen an den Agenten vorzunehmen, weisen die strukturellen Adaptionsprozesse sogenannte *Adaptation Enactments* an, die Agenten im Hinblick auf lokale Performanz-Indikatoren, die Auskunft über die Güte der selbstorganisierenden Dynamik geben, zu observieren. In beiden Fällen, einer strukturellen Adaption der Koordinationsprozesse und einer parametrischen Adaption der Koordinationsprozessparameter, ist es wichtig, dass die Agenten einen Konsens darüber finden, ob eine Adaption durchgeführt werden soll oder nicht. Eine mögliche Lösung dazu ist ein verteilter Wahl-Algorithmus, wie er in Abschnitt 4.4.4 beschrieben wird.

Der Entwicklungsansatz der strukturellen Adaption von Koordinationsprozessen orientiert sich eng an dem existierenden Entwicklungsansatz aus dem *SodekoVS* Projekt [SBP+09] (vgl. dazu auch Abschnitt 3.2.3). In Abbildung 3.7 wurde dabei das iterative Vorgehen aus der *SodekoVS*-Methodologie veranschaulicht, das in fünf Schritten einen Top-Down-Ansatz beschreibt. Der erste Schritt des SodekoVS-Entwicklungsansatzes besteht aus der Anforderungsphase, in der das gewünschte Systemverhalten spezifiziert wird. In der folgenden Analysephase werden dazu passende Koordinationsmetaphern identifiziert und so kombiniert, dass sie das gewünschte, adaptive Verhalten hervorbringen. In der Design-Phase werden aus diesen Metaphern die entsprechenden Koordinationsstrategien für die Realisierung abgeleitet. Für die einzelnen Koordinationsstrategien existieren generische Implementationen in Software-Bibliotheken, die bei der Realisierung des Systems verwendet werden können. In der Test-Phase wird das System schließlich durch Simulation validiert. Dabei wird das Systemverhalten, das bei der Simulation auftritt, beobachtet und mit dem in den Anforderungen spezifizierten Verhalten verglichen.

Wenn entweder im Rahmen dieser Validierung oder als Ergebnis der initialen Analyse beobachtet wird, das bestimmte Koordinationsprozesse oder bestimmte Koordinationsprozesskonfigurationen unter bestimmten Bedingungen zwar das vorgesehene Systemverhalten realisieren, unter anderen Bedingungen aber zu einem fehlerhaften oder ineffektiven Verhalten führen, dann

sind dieses mögliche Bedingungen, bei denen der Einsatz von strukturellen Adaptionen der Koordinationsprozesse oder ihrer Parameter sinnvoll ist. Die Herausforderung hierbei ist es, genau diese Bedingungen zu identifizieren, die eine strukturelle oder parametrische Adaption erfordern und daraus folgend ihre Abbildung auf Performanz-Indikatoren, die Aussagen über die Güte der selbstorganisierenden Dynamik geben. Eine Untersuchung über geeignete Performanz-Indikatoren, die Aussagen über die Güte von selbstorganisierenden Dynamiken geben, findet sich in [VL12]. Als Ergänzung zu der *SodekoVS*-Methodologie werden im Rahmen dieser Dissertation antizipierte strukturelle Adaptionen vorgeschlagen. Basierend auf dem iterativen Vorgehen, das bei dem Entwurf und der Realisierung der Koordinationsprozesse verwendet wird, sollte das Design und die Realisierung der strukturellen Adaptionen ebenfalls in iterativer Weise geschehen. Dabei sollten die Bedingungen, unter denen eine strukturelle Adaption ausgeführt wird, in der Anforderungs- und Analysephase erhoben und im Rahmen einer Simulations-basierten Validierung weiter verfeinert werden.

Wie eingangs in diesem Abschnitt beschrieben besteht eine konzeptionelle Ähnlichkeit zwischen dem lokalen Planen eines BDI-Agenten und der Auswahl einer geeigneten strukturellen Adaption. Ein BDI-Agent entscheidet auf Basis seiner Beliefs, die Auskunft über sein aktuelles Wissen bzw. seinen Zustand geben, darüber welche Ziele (Goals) er verfolgt. Dabei werden die Beliefs in den Bedingungen der Ziele referenziert. Um diese Ziele zu erreichen, stehen dem Agenten dabei eine Reihe von vordefinierten Plänen mit Handlungsanweisungen zur Verfügung. Über eine entsprechende Planungskomponente, die Teil eines jeden BDI-Agenten bzw. der zugehörigen Agentenplattform ist, wird nun ausgewählt, welche Pläne geeignet sind, um die Ziele zu erreichen. Die strukturelle Adaption der Koordinationsprozesse funktioniert in ähnlicher Weise. Die beteiligten Komponenten verfügen über eine Reihe von Performanz-Indikatoren, die Aussagen über die Güte des selbstorganisierenden Verhaltens geben. Diese Indikatoren werden von den Bedingungen der Adaptionen referenziert. Auf Basis dieser Bedingungen kann eine Komponente feststellen, dass die selbstorganisierende Dynamik aus Sicht der lokalen Komponente heraus nicht mehr das gewünschte Verhalten zeigt. Dadurch wird dann die verteilte Konsensfindung ausgelöst, mit dem Ziel, einen verteilten Konsens unter den betroffenen Komponenten im System darüber zu finden, welcher bzw. ob ein vordefinierter Adaptionsplan ausgewählt wird. Der Adaptionsplan enthält dann entsprechende Anweisungen, in welcher Art die strukturelle Anpassung an den Koordinationsprozessen erfolgen soll, um somit die selbstorganisierende Dynamik zu beeinflussen. Diese konzeptionelle Nähe der beiden Absätze ist in Abbildung 4.10 dargestellt.

### 4.4.2. Architekturerweiterung für strukturelle Adaptionen

Da die strukturelle Adaption von Koordinationsprozessen sowohl konzeptionell als auch methodisch eng an den *SodekoVS*-Ansatz geknüpft ist, handelt

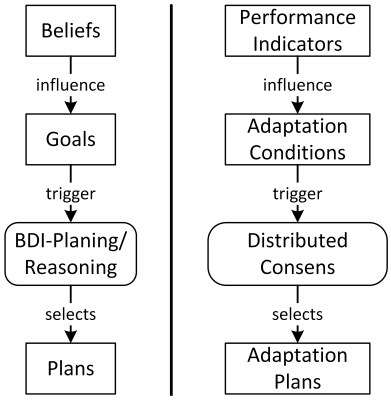



**Abbildung 4.10.:** Konzeptioneller Vergleich zwischen Planen in einem BDI-Agenten und der strukturellen Adaption.

es sich bei der entsprechenden Architektur dementsprechend auch um eine Erweiterung der *SodekoVS*-Referenzarchitektur (vgl. Abbildung 3.8 und 3.9). Die Erweiterung umfasst dabei primär die Einführung sogenannter *Adaptation Enactments*, die (ähnlich den *Coordination Enactments*, welche Agenten mit der Fähigkeit zur Koordination ausstatten) Systeme in die Lage versetzen, strukturelle Adaptionen dynamisch zur Laufzeit vorzunehmen. Dargestellt ist diese Erweiterung der *SodekoVS*-Referenzarchitektur in Abbildung 4.11. Die *Adaptation Entactments* sind dabei Teil der Koordinationsschicht und daher unabhängig von der eigentlichen Logik der Anwendung. Ihre Aufgabe ist es, ähnlich wie die der *Coordination Enactments*, die Agenten zu observieren. Allerdings beeinflussen sie im Gegensatz zu den *Coordination Enactments* nicht die Agenten, sondern die Koordinationsprozesse. Dabei werden Teilstrukturen verändert, indem die entsprechenden Koordinationsmedien, die diese Strukturen auf technischer Ebene umsetzen, aktiviert bzw. deaktiviert werden oder, sofern es sich um eine parametrische Adaption handelt, indem die Koordinationsparameter, die die Medien parametrisieren, verändert werden. Ein *Adaptation Enactment* besteht auf zwei Komponenten (siehe Abbildung 4.11):

- Einem *Monitor*, dessen Aufgabe es ist, die Performanz-Indikatoren, die Auskunft über die Qualität des selbstorganisierenden Verhaltens geben, zu observieren. Wie bereits erwähnt, sind diese Performanz-Indikatoren Teil der Agenten und geben daher deren lokale Sichtweise auf die Qualität des selbstorganisierenden Verhaltens wieder. Wenn der *Monitor* hier eine abfallende Performanz detektiert, initiiert er den Abstimmungsprozess über die entsprechende, vordefinierte strukturelle Adaption.

- Als Schnittstelle für den verteilten Abstimmungsprozess wird eine *Service* Komponente verwendet. Wenn der *Monitor* einen Abfall der Performanz des selbstorganisierenden Verhaltens detektiert, initiiert er den verteilten Abstimmungsprozess und übernimmt dabei die Rolle des Leiters für den Prozess. Dabei verwendet er die *Service* Schnittstelle der an-

deren Agenten, um diesen die geplante Adaption zur Wahl zu stellen. Basierend auf ihren lokalen Informationen bzw. ihrem lokalen Zustand entscheiden die aufgerufenen Agenten darüber, ob sie der Adaption zustimmen oder nicht. Der Leiter analysiert folglich die erhaltenen Abstimmungsresultate und überprüft, ob eine benötige Mehrheit erzielt wurde. Wenn dieses der Fall ist, wird die vorgeschlagene Adaption ausgeführt, indem die betroffenen Koordinationsmedien aktiviert oder deaktiviert werden bzw. die entsprechenden Koordinationsparameter der Koordinationsmedien angepasst werden.

Da eine strukturelle Adaption eine Anpassung der Koordinationsmedien bedeutet, wurde eine Erweiterung dieser um entsprechende Schnittstellen zur Aktivierung und Deaktivierung bzw. zur Veränderung von Koordinationsparametern zur Laufzeit nötig [PRD15].

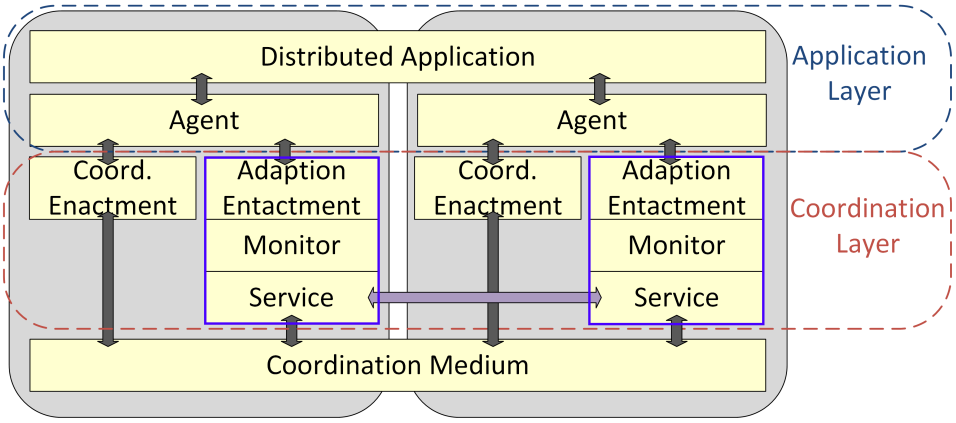



**Abbildung 4.11.:** Erweiterung der SodekoVS Referenzarchitektur um strukturelle Adaptionen (blaue Rechtecke) - nach (PRD15, S. 5).

### 4.4.3. Beschreibung von strukturellen Adaptionen

Das Konzept der strukturellen Adaptionen basiert auf einem antizipierenden Ansatz. Das bedeutet, dass die möglichen Adaptionen bereits zur Entwurfszeit von einem Entwickler beschrieben werden, um dann zur Laufzeit automatisiert von der Ausführungsumgebung ausgeführt zu werden, wenn entsprechende Bedingungen eingetreten sind, die eine strukturelle Adaption nötig machen. Daher wird eine Beschreibungssprache benötigt, die es ermöglicht, sowohl die Bedingungen unter denen eine solche Adaption ausgeführt werden soll als auch die Art der Adaption zu beschreiben. Da das Konzept der strukturellen Adaption konzeptionell auf dem *SodekoVS* Ansatz [SBP+09] und implementationstechnisch auf der *Coordination Space* Ausführungsumgebung [VSL+10] basiert, wurde die *MASDynamics* Sprache zur Beschreibung von selbstorganisierenden Verhalten in Multiagentensystemen [SR09b] als Ausgangsbasis verwendet und um die entsprechenden Möglichkeiten erweitert [PR15, PRD15]. Dazu wurden zunächst die bereits

existierenden Möglichkeiten zur Beschreibung von Koordinationsprozessen erweitert, sodass festgelegt werden kann, ob diese zur Startzeit aktiv oder inaktiv sein sollen. Dadurch wird es Entwicklern ermöglicht, zur Entwurfszeit mehrere Koordinationsprozesse mit unterschiedlichen Verhalten zu definieren, von denen nur eine Untermenge zum Start der Anwendung aktiv sind.

Der erste Teil der Spracherweiterung bezieht sich dabei auf die Agenten, die an dem Adaptionsprozess teilnehmen. Dadurch wird die Möglichkeit gegeben, entweder alle vorhandenen Typen von Agenten an diesem Prozess teilnehmen zu lassen oder nur eine ausgewählte Untermenge. Letzteres kann sinnvoll sein, wenn nur eine Untermenge der Agenten tatsächlich Anteil an dem selbstorganisierenden Verhalten der Anwendung hat. Resultierend aus dieser Beschreibung werden vom Koordinationsrahmenwerk zum Start der Anwendung *Adaptation Enactments* für die entsprechenden Typen von Agenten erstellt, sodass diese den lokalen Zustand der Agenten im Hinblick auf das selbstorganisierende Verhalten observieren können, um ggf. den strukturellen Adaptionsprozess auszulösen.

Der zweite Teil der Beschreibung beschäftigt sich mit den eigentlichen Adaptionen. Jede Adaption ist zunächst mit einem eindeutigen Identifikator (`id`) beschrieben. Des Weiteren kann ein Flag (`reset`) spezifiziert werden, das an gibt, ob der Performanz-Indikator, der dieser Adaption zugeordnet ist, nach einem fehlgeschlagenen Adaptionsversuch zurückgesetzt werden soll. Damit ist es möglich, zu verhindern, dass wiederholte Adaptionsversuche das System überfluten, wenn nur eine Untermenge von Agenten einem ineffizienten selbstorganisierenden Verhalten ausgesetzt ist und somit die Mehrheit des Systems im Hinblick auf das selbstorganisierende Verhalten gut funktioniert und daher einer vorgeschlagenen Adaption nicht zustimmen wird. Weiterhin ist es möglich, eine Adaption für eine bestimmte Zeit zu Beginn der Anwendung zu blockieren (`startDelay`). Dadurch können Oszillationsprobleme vermieden werden, die auftreten können, wenn es eine Weile dauert, ehe sich das selbstorganisierende Verhalten einer Anwendung in zielgerichteter Weise einstellt und somit eine verfrühte Adaption der Koordinationsprozesse kontraproduktiv wäre.

Die weiteren Informationen, die in der Beschreibungssprache hinterlegt sind, betreffen den Abstimmungsprozess. Dabei kann zunächst spezifiziert werden, wie viele Antworten (`answers`) der Leiter des Abstimmungsprozesses erwartet, ehe er mit der Auswertung beginnt. Ein Quorum (`quorum`) gibt dabei an, wie viel Prozent der Agenten einer vorgeschlagenen Adaption mindestens zustimmen müssen, damit diese ausgeführt wird. Außerdem ist es möglich, eine Zeitsperre (`timeout`) anzugeben, nach deren Ablauf der Abstimmungsleiter mit der Auswertung der erhaltenen Antworten beginnt, selbst wenn er zu diesem Zeitpunkt noch nicht alle benötigten Antworten erhalten hat. Damit soll vermieden werden, dass durch das Ausbleiben von Antworten eine mögliche Adaption verhindert wird.

Weitere Informationen bezüglich des Adaptionsprozesses beziehen sich auf die Anpassungen an den betroffenen Koordinationsprozessen. Dabei wird

zunächst der Identifikator des Koordinationsmediums, welches den Koordinationsprozess abbildet, angegeben (`realization_id`). Anschließend geht es darum, wie das betroffene Koordinationsmedium angepasst werden soll. Über ein Flag kann entsprechend gesteuert werden, ob das betroffene Medium aktiviert oder deaktiviert werden soll (`active`). Wenn sich an dem Aktivierungszustand des Mediums nichts ändern soll, sondern stattdessen ein Koordinationsparameter angepasst wird, kann dieser entsprechend mit neuen Wert angegeben werden (`parameter` und `value`).

Schließlich werden noch die Informationen benötigt, die den lokalen Zustand des Agenten beschreiben, unter denen dieser die strukturelle Adaption vorschlägt (`constraints`). Diese lokalen Bedingungen bilden dabei den Performanz-Indikator ab, der basierend auf der lokalen Sicht des Agenten die Qualität des selbstorganisierenden Verhaltens beschreibt. Dabei muss zunächst angegeben werden, welches Element (`element`) des Agenten als Performanz-Indikator verwendet wird. Da die Ausführungsumgebung des *Coordination Space*, auf dem BDI-Agenten Konzept basiert, muss es sich bei diesem Element um einen *Belief* des Agenten handeln. Neben dem Element werden eine Kondition (`condition`) und ein Schwellwert (`threshold`) angegeben. Die Kondition beschreibt dabei die Bedingung, unter der im Agenten ein signifikanter Qualitätsverlust des selbstorganisierenden Verhaltens festgestellt wird, was schließlich dazu führt, dass der Agent den Adaptionsprozess auslöst. Der Schwellwert bezieht sich auf das gleiche Element wie die Kondition, ist dabei allerdings nicht so strikt wie diese formuliert. Der Schwellwert wird von dem Agenten herangezogen, wenn dieser über seinen *Adaptation Enactment* Service die Adaption zur Durchführung vorgeschlagen bekommt. Wenn die Schwellwert-Bedingung dabei zutrifft, wird der Agent der vorgeschlagenen Adaption zustimmen. Der Schwellwert bildet somit einen negativen Trend ab, der dem Agenten erlaubt, eine unzulängliche Performanz des selbstorganisierenden Verhaltens zu antizipieren, sodass er einer vorgeschlagenen Adaption zustimmt, sollte ein anderer Agent diesem unzulänglichen Verhalten bereits ausgesetzt sein und daher den Adaptionsprozess ausgelöst haben.

Quelltext A.1 zeigt die XML Schema Definition (XSD) der Adaptionserweiterung für die *MASDynamics* Sprache. In Abschnitt 7.1.1 wird mit der *Mars World* ein Anwendungsbeispiel vorgestellt, in dem u.a. die hier beschriebenen Elemente der *MASDynamics* Spracherweiterung anhand von konkreten Werten weiterführend erläutert werden.

### 4.4.4. Adaptions- und Abstimmungs-Prozess

Der Adaptions- und der zugehörige Abstimmungs-Prozesses wird initiiert, sobald sich der Wert eines Agenten *Beliefs* ändert, der in der entsprechenden Adaptionsbeschreibung als lokaler Performanz-Indikator für das selbstorganisierende Verhalten spezifiziert wurde. Dieser Wert wird von der *Monitor* Komponente im *Adaptation Enactment* des Agenten auf die entsprechende

Bedingung aus der Adaptionsbeschreibung hin überprüft. Wenn die Bedingung erfüllt ist und aktuell kein Adaptionsprozess mit einer höheren Priorität aktiv ist, wird der zugehörige Adaptionsprozess gestartet. Die Priorität eines Adaptionsprozesses ergibt sich aus dem *Hash*-Wert[3] des Identifikators des initiierenden Agenten. Aufgrund der geringen Wahrscheinlichkeit von Kollisionen bei *Hash*-Werten können diese als eindeutige Identifikatoren in verteilten Systemen ohne globales Wissen verwendet werden. Dadurch, dass überprüft wird, ob kein Adaptionsprozess mit einer höheren Priorität aktiv ist, kann sichergestellt werden, dass nur ein Prozess zurzeit aktiv ist. So ist es möglich, Interferenzen zwischen diesen Prozessen zu vermeiden. Sollte also die Adaptions-Bedingung erfüllt und kein anderer Prozess mit einer höheren Priorität aktiv sein, ruft der *Monitor* die entfernten Abstimmungsdienste der anderen Agenten auf. Anschließend wartet er, bis er entweder die benötige Anzahl an Antworten erhalten hat oder, sofern ein Timeout spezifiziert wurde, dieser abgelaufen ist.

Der Abstimmungsdienst, der wie der *Monitor* Teil des *Adaptation Enactment* eines Agenten ist (siehe Abbildung 4.11), überprüft, wenn ihm eine Adaption zur Abstimmung vorgeschlagen wird, zunächst, ob ihm bereits eine andere Adaption mit höherer Priorität vorgeschlagen wurde. Wenn ja, wird er, um Interferenzen zu vermeiden, die Adaption mit der niedrigeren Priorität ablehnen. Wenn dieses nicht der Fall ist, überprüft er seinen lokalen *Belief*, der in der Adaptionsbeschreibung referenziert wird, gegen den ebenfalls dort spezifizierten Schwellwert. Sollte der Schwellwert erreicht sein, wird er den negativen Trend im Bezug auf das selbstorganisierende Verhalten erkennen und entsprechend für die vorgeschlagene Adaption stimmen.

Der *Monitor*, der ursprünglich die Adaption vorgeschlagen hat, beginnt, sobald er alle erwartenden Antworten erhalten hat oder das Timeout abgelaufen ist, diese auszuwerten. Dabei überprüft er, ob die in der Adaptionsbeschreibung hinterlegte Mehrheit erreicht wurde. Sollte das Timeout eingetreten sein, bevor alle benötigten Antworten erhalten wurde, oder keine Mehrheit erreicht worden sein, so wird der Adaptionsvorgang abgebrochen. Dieses geschieht auch, wenn der Adaptionsprozess an dieser Stelle von einem anderen Prozess mit einer höheren Priorität unterbrochen wurde. Wenn keine der Abbruchbedingungen eingetreten ist, wird die strukturelle Adaption ausgeführt werden. Dieses geschieht dadurch, dass der *Monitor* die betroffenen selbstorganisierenden Prozesse beeinflusst, indem über die entsprechenden Schnittstellen die Koordinationsmedien, die dem betroffenen Koordinationsprozess zugeordnet sind, aktiviert bzw. deaktiviert werden oder die betroffenen Koordinationsparameter anpasst werden. Dieser Ablauf ist entsprechend in Form eines UML Aktivitätsdiagramm in Abbildung 4.12 dargestellt.

[3] *Hash*-Wert ist ein Begriff aus der Computertechnik im Bereich der Kryptologie und bezeichnet einen alphanumerischen Wert, der durch eine besondere Form der *Hash*-Funktion erzeugt wird. Die Besonderheit an dieser mathematischen Funktion ist, dass sie eine beliebig lange Zeichenfolge auf eine Zeichenfolge mit fester Länge abbildet.

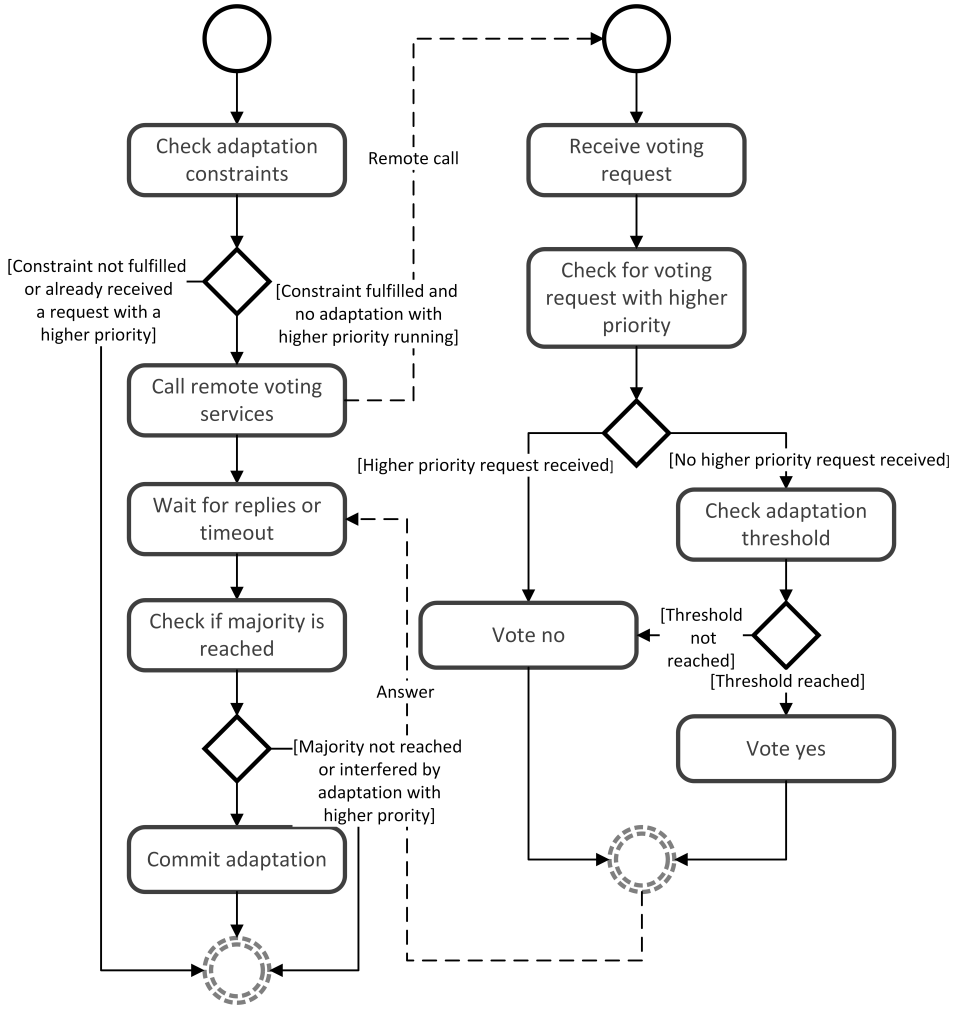



**Abbildung 4.12.:** UML Aktivitätsdiagramm das den Adaptions- und Wahl-Algorithmus im Monitor (links) und Abstimmungsdienst (rechts) veranschaulicht - nach (PRD15).

## 4.5. Zusammenfassung

In diesem Kapitel wurde die Entwicklung eines Konzepts zur systematischen Konstruktion selbstorganisierender Systeme mit dem speziellen Fokus auf selbstorganisierenden Multiagentensystemen vorgestellt und um ein neuartiges Konzept zur strukturellen Adaption von Koordinationsprozessen erweitert. Dazu wurden zu Beginn des Kapitels zunächst die speziellen Herausforderungen, die die Konstruktion selbstorganisierender Systeme bedingen, herausgearbeitet. Darauf aufbauend wurden existierende Ansätze im Hinblick auf ihre Fähigkeit untersucht, dem erstellten Anforderungsprofil gerecht zu werden. Dafür wurde eine allgemeine Klassifizierung existierender Ansätze vorgenommen, um zu verdeutlichen, welche generellen Lösungsansätze für diese Fragestellungen existieren. Dabei wurden die folgenden Klassen identifiziert: Integrierte Agentenumgebungen, generische Umgebungsinfrastrukturen und spezialisierte Koordinationsrahmenwerke. Im Rahmen dieser Untersuchung wurden die jeweiligen Stärken und Schwächen der identifizierten Klassen deutlich. Insgesamt ließ sich dabei ableiten, dass keine der untersuchten Arbeiten in ihrer Grundform in der Lage ist, das Anforderungsprofil vollständig zu erfüllen. Insbesondere fehlte den Ansätzen eine Trennung zwischen Kern- und Koordinationsfunktionalität sowie die nötige Flexibilität und Wiederverwendbarkeit in Bezug auf die Anwendungsdomäne und die Nutzbarkeit von Koordinationsmechanismen als Abstraktion der zugrundeliegenden Koordinationsmechaniken.

Ausgehend von der Analyse existierender Ansätze wurde das Konzept des *Coordination Space* präsentiert. Dieses Konzept unterstützt die Konstruktion selbstorganisierender Systeme auf zwei Arten: Durch eine Integrationsarchitektur und durch eine korrespondierende Ausführungsumgebung. Die Integrationsarchitektur verdeutlicht dabei auf konzeptioneller Ebene die lose Kopplung zwischen Kern- und Koordinationsfunktionalität und ermöglicht es, durch die Verwendung von Koordinationsmedien, Selbstorganisationsfunktionalität dediziert zu modellieren und zu implementieren. Dadurch ist es möglich, Selbstorganisationsfunktionalität als Aspekt erster Ordnung bei der Konstruktion selbstorganisierender Systeme zu nutzen. Die Integrationsarchitektur basiert auf der *SodekoVS*-Referenzarchitektur und entwickelt diese weiter. Der *Coordination Space* leitet dabei die grundlegende Trennung von Kern- und Koordinationsfunktionalität an und ist dabei sowohl unabhängig von der Anwendungsdomäne als auch auch von der realisierten Selbstorganisationsfunktionalität. Die im *Coordination Space* enthaltenen Koordinationsmedien kapseln dabei die austauschbaren und spezifischen Aspekte der Anwendungen. Die praktische Nutzbarkeit der Integrationsarchitektur wird durch die zugehörige Ausführungsumgebung realisiert. Die Ausführungsumgebung verdeutlicht dabei, wie Selbstorganisationsfunktionalität nahtlos in die Multiagentenplattform *Jadex* eingebunden werden und dort als integraler Bestandteil zur Implementation verteilter Systeme genutzt werden kann. Die Konstruktion der eigentlichen Kernfunktionalität kann

dabei unabhängig von der Selbstorganisationsfunktionalität erfolgen, sodass letztere als orthogonaler Bestandteil verstanden werden kann.

Ein weiterer wichtiger Aspekt, der in diesem Kapitel thematisiert wurde, ist das Thema der Verteilung des *Coordination Space* über die Grenzen von Plattformen und Netzwerkknoten hinweg. Damit wurden ursprünglich bestehende Limitierungen der zugrundeliegenden Architekturen und Rahmenwerke überwunden. Dazu wurde auf Ebene der Integrationsarchitektur die Komponente des *Medium Endpoint* eingeführt. Auf der Ebene der Ausführungsumgebung wurde u.a. die *CoordinationContextID* eingeführt, die es ermöglicht, verteilte Anwendungen, die auf unterschiedlichen Plattformen und Netzwerkknoten ausgeführt werden, einem gemeinsamen Koordinationskontext zuzuordnen. Mit diesen Erweiterungen des *Coordination Space* Konzepts ist eine transparente Nutzung verteilter Koordinationsfunktionalität möglich, die intern den Aufbau beliebiger Netzwerktopologien unterstützt. Ferner wurde in diesem Kapitel die Flexibilität und Wiederverwendbarkeit von Koordinationsmedien gezeigt. Hierzu wurden exemplarisch mehrere verteilte Koordinationsmedien konstruiert.

Schließlich wurde das Konzept der strukturellen Adaption von Koordinationsprozessen präsentiert. Dieses Konzept ermöglicht zur Laufzeit den dynamischen Austausch bzw. die Rekonfiguration der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden. Damit ist es möglich, die Struktur der Selbstorganisationsfunktionalität anzupassen, sofern diese zur Laufzeit zu einem emergenten Fehlerverhalten führt oder in anderer Form von negativen Auswirkungen betroffen ist, die wiederum von internen oder externen Einflüssen verursacht werden können. Auf konzeptioneller Ebene ist das Konzept der strukturellen Adaption eine Erweiterung der *SodekoVS*-Referenzarchitektur, auf technischer Ebene wird die Ausführungsumgebung des *Coordination Space* erweitert, sodass es einerseits möglich ist, Koordinationsmedien, die die technische Realisierung der Koordinationsprozesse darstellen, dynamisch zur Laufzeit in Form von Parameteränderungen oder Aktivierung bzw. Deaktivierung zu beeinflussen. Andererseits wurde eine entsprechende Infrastruktur als Erweiterung der *Coordination Space* Ausführungsumgebung geschaffen, die es den betroffenen Agenten ermöglicht, einen verteilten Konsens über mögliche Adaptionen der Koordinationsprozesse zu finden.

# 5. Skalierbare, verteilte und adaptive Co-Simulationssysteme

Dieses Kapitel gibt einen Überblick über skalierbare, verteilte und adaptive Simulationssysteme. Simulationen spielen nicht nur eine entscheidende Rolle, wenn es um die Validierung und Evaluation selbstorganisierender Systeme als Teil eines iterativen Entwicklungsprozesses geht, sondern profitieren ihrerseits auch von selbstorganisierenden Eigenschaften oder der Fähigkeit zur Koordination. Von besonderem Interesse sind in diesem Zusammenhang Co-Simulationen (siehe [GKL06]). Bei einer Co-Simulation formen unterschiedliche Sub-Systeme zusammen ein gekoppeltes Problem. Dabei erfolgt sowohl die Modellierung als auch die Ausführung der Simulation verteilt. Unter einer verteilten Modellierung wird in diesem Zusammenhang die Verwendung von unterschiedlichen Werkzeugen, Programmen oder Programmiersprachen verstanden. Während der verteilten Ausführung interagieren die einzelnen Sub-Simulationen, indem sie Daten untereinander austauschen. Die Interaktion erfolgt dabei i.d.R. über im Vorfeld definierte Schnittstellen. Die unterschiedlichen Sub-Systeme bzw. -Simulationen erscheinen untereinander dabei jeweils als *„Black-Box“*, d.h. sie haben keine Informationen über den internen Aufbau der anderen Systeme, sondern nehmen diese nur über die entsprechenden Eingaben und Ausgaben der definierten Schnittstellen wahr. Eine der Hauptherausforderungen bei der Realisierung solcher Co-Simulationssysteme ist die sinnvolle Orchestrierung bzw. Koordination der enthaltenen Systeme. Zu diesem Zweck wird im nächsten Kapitel (6) die Entwicklung eines Rahmenwerks behandelt, das die Koordination von Simulationskomponenten bzw. die Realisierung von selbstorganisierenden Eigenschaften in Co-Simulationssystemen ermöglicht. Dazu wird ein generisches Rahmenwerk präsentiert, das die Koordination von heterogenen Komponenten, unterstützt. In diesem Kapitel geht es nun zunächst darum, ein wichtiges Anwendungsgebiet für Co-Simulationen vorzustellen.

Ein aktuell wichtiges Anwendungsgebiet für Co-Simulation ist die Planung und Realisierung sogenannter *Smart Cities* (vgl. [PBDR14b]). *Smart City* ist dabei ein Sammelbegriff für gesamtheitliche Entwicklungskonzepte, die darauf abzielen, Städte effizienter, technologisch fortschrittlicher und umweltverträglicher zu gestalten. Der Begriff wird seit den 2000er Jahren von unterschiedlichen Akteuren aus Politik, Wirtschaft, Verwaltung und Stadtplanung verwendet, um technologiebasierte Veränderungen und Innovationen in urbanen Räumen zusammenzufassen. Dabei werden verschiedene Charakteristika zum Zweck des Vergleichs von Städten im Bereich *„Smartness“* definiert: *Smart Economy* (Wirtschaft), *Smart People* (Bevölkerung), *Smart Mobility*

(Mobilität), *Smart Environment* (Umwelt), *Smart Living* (Leben) und *Smart Grid* (Energienetze).

In diesem Kapitel wird nun zunächst ein Überblick darüber gegeben, wie die Interoperabilität von Simulationen dazu beitragen kann, Fragestellungen im Umfeld der *Smart Cities* zu beantworten. Nachdem einige Ansätze zur Herstellung von Interoperabilität von Simulationen vorgestellt wurden, wird exemplarisch eine Servicekomponenten-basierte Architektur präsentiert, die es ermöglicht, adaptive, verteilte Co-Simulationssysteme zu realisieren. Basierend auf dieser Architektur wird ein Beispiel für die skalierbare Integration von 4GL[1]-Modellen im Umfeld der *Smart Grid* Simulationen präsentiert. Schließlich wird mit dem *Simulationen als ein Service* Konzept auf einen Ansatz eingegangen, der die Realisierung von Co-Simulationen auf Basis von Cloud Computing (siehe [Sos11]) Technologien unterstützt.

## 5.1. Interoperabilität von Simulationen

Sowohl die Planung und Realisierung als auch der Betrieb einer intelligenten Stadt (*Smart City*) bedingen die Berücksichtigung vieler komplexer Faktoren und Akteure. Um deren Zusammenwirken in Hinblick auf die Zielsetzungen während aller Entwicklungsphasen gewährleisten zu können, ist es unerlässlich, Simulationen durchzuführen. Die ganzheitliche Betrachtung einer intelligenten Stadt erfordert dabei unterschiedlichste Simulationskonzepte. Unabhängig von der Domäne gilt es beispielsweise, Betriebskonzepte und deren Wirtschaftlichkeit zu prüfen und ggf. zu optimieren (z.B. [LSS12, RGS+13]). Während hierfür feingranulare Simulationen unter Berücksichtigung mikroskopischer Aspekte benötigt werden, so z.B. bei der Regelung, sind für Planungsfragen in Bezug auf Stadt- und Netzentwicklung i.d.R. makroskopische Simulationen von Bedeutung. Abbildung 5.1 stellt gängige Typen von Simulationen mit Bezug auf die Energieversorgungsplanung im Kontext des Smart Grids dar und setzt sie in einen zeitlichen Kontext. Außerdem sind die verschiedenen Simulations- und Planungsebenen und deren Verzahnungen aufgeführt. Neben den Zeitskalen unterscheiden sich diese unterschiedlichen Simulationstypen auch in ihren Anforderungen, die sich aus der Fachlichkeit und dem Level der Abstraktion (mikroskopisch oder makroskopisch) ergeben. Natürlich gibt es auch Mischformen der Simulation, in denen mikroskopische und makroskopische Aspekte kombiniert werden. In solchen Fällen spricht man von mesoskopischen Simulationen (siehe z.B. [RPS12]). Bei der Simulation von mikroskopischen Vorgängen können aufgrund der hohen Komplexität meist langfristige Effekte nicht untersucht werden, umgekehrt ist die detaillierte Betrachtung von Grenzfällen in makroskopischen Simulationen aufgrund des höheren Abstraktionslevels oftmals nicht möglich. So können Simulationen zur Stadtentwicklung Auswirkungen auf mikroskopische Faktoren i.d.R. nicht unmittelbar abbilden

[1] 4GL steht für „*fourth generation languages*“, also Programmiersprachen der vierten Generation. Beispiele für solche Sprachen sind u.a.: MatLab, Modelica und GAMS.

und lassen somit keine zuverlässige Abschätzung zu. Für die ganzheitliche Planung von *Smart Cities* ist daher die Betrachtung sowohl gesamtstädtischer Aspekte als auch die Einbeziehung der Besonderheiten einzelner Stadtteile und Liegenschaften von besonderer Bedeutung. Auf technischer Ebene bedeutet dies, dass die unterschiedlichen Simulationskonzepte für eine ganzheitliche Betrachtung zu einem Co-Simulationssystem kombiniert werden müssen. Im Folgenden werden daher einige existierende Ansätze vorgestellt, welche die Interoperabilität von Simulationen unterstützen.

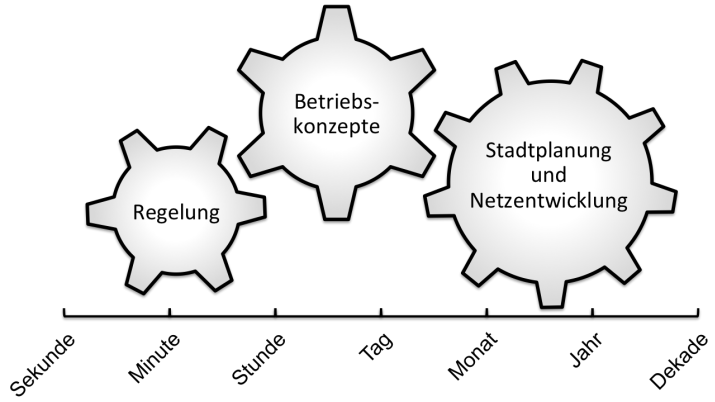



**Abbildung 5.1.:** Zeithorizont und Planungsebenen in unterschiedlichen Simulationstypen im Bereich der Energieversorgung - aus (PBDR14b, S. 2).

### 5.1.1. Ansätze zur Interoperabilität von Simulationen

Für Simulationen im Allgemeinen und für *Smart Grid* Anwendungen im Speziellen existieren eine Vielzahl von unterschiedlichen Ansätzen und Werkzeugen. Insbesondere im militärischen Anwendungsbereich wurden dabei in der Vergangenheit größere Fortschritte auf dem Gebiet der Simulationswerkzeuge und Interoperabilität erzielt. Das in den früheren 1990ern begonnene Bestreben, einen Standard für Simulationsinteroperabilität zu schaffen, resultierte in dem *High Level Architecture* (HLA) IEEE Standard für Modellierung und Simulationen [IEE00]. Die HLA beschreibt eine technische Architektur, die entwickelt wurde, um verschiedene Simulationssysteme und -Module interoperabel und wiederverwendbar zu machen. Es handelt sich um ein Rahmenwerk, mit dem Entwickler ihre Simulationen abstrahieren und beschreiben können, um die Interoperabilität mit anderen Simulationskomponenten herzustellen. Allerdings wird dieser Ansatz aufgrund seiner hohen Komplexität außerhalb des militärischen Anwendungsbereichs kaum genutzt (vgl. [BdBV08]).

Ein spezifischer Ansatz zur Simulation von integrierten Energiesystemen und Kontrollstrukturen wurde für das Department of Energy[2] (DOE) am *Pacific Northwest National Laboratory* (PNNL) entwickelt [CSG08]. *GridLAB-D*

[2]Energieministerium der Vereinigten Staaten von Amerika

erlaubt dabei die Spezifikation einer Reihe von unterschiedlichen Simulationsszenarien und die Simulation von Millionen von unabhängigen Geräten. Im Bezug auf die Skalierbarkeit verfügt es daher über sehr gute Eigenschaften. Allerdings wird nur die Einbindung von externen *MATLAB*[3] Modelle und die Anbindung an die quelloffene *MySQL*[4] Datenbank unterstützt. Daher ist die Interoperabilität von Simulationen auch nur eingeschränkt unterstützt.

Das *Mosaik* [SS12, RWM+14] Rahmenwerk konzentriert sich, verglichen mit *GridLAB-D*, deutlich mehr auf die Einbindung heterogener Simulationskomponenten und stellt die Wiederverwendbarkeit der Module sicher. Der Fokus von *Mosaik* liegt dabei auf der Realisierung von skalierbaren *Smart Grid* Anwendungssimulationen. Diese werden durch das Zusammenwirken folgender vier Komponenten ermöglicht:

1. Eine programmiersprachenunabhängige Programmierschnittstelle, die die Integration von bestehenden Simulationen, Simulationsmodellen und Strategien in das Mosaik Rahmenwerk erlaubt.

2. Eine Programmierschnittstelle zur Definition von Szenarien, die zur Beschreibung der skalierbaren Simulationen dient. In dieser können die unterschiedlichen Simulationsprozesse und -modelle definiert werden. Darüber hinaus können die Verbindungen zwischen den einzelnen Elementen beschrieben werden.

3. Ein Simulationsmanager, der als zentrale Komponente die Ausführung der Simulation handhabt.

4. Eine ereignisbasierte, diskrete Ausführungsumgebung, welche die Ausführung aller angeschlossenen Simulationsmodule koordiniert. Simulationsmodule können dabei unterschiedliche zeitliche Auflösungen realisieren, die ferner auch noch zur Laufzeit verändert werden können.

### 5.1.2. Servicekomponenten-basierte Architektur für Simulationssysteme

Um sowohl mikroskopische als auch makroskopische Simulationen im Rahmen einer Co-Simulation zu verbinden, bedarf es einer integrativen Architektur, um die unterschiedlichen Simulationskonzepte, die oftmals auf unterschiedlichen Zeitskalen und -modellen sowie Werkzeugen basieren, zu verbinden. Eine solche Architektur sollte in der Lage sein, Simulationen auf Makroebene durchzuführen und im Bedarfsfall auf Komponenten zurückzugreifen, die den Detaillierungsgrad bzw. Abstraktionsgrad dynamisch anpassen. Eine entsprechende Architektur, der die Überlegung zu Grunde liegt, das unterschiedliche Simulationen als Dienste (Engl. Services) eines Informations- und Simulationssystems angeboten werden, wurde im Rahmen dieser Dissertation entwickelt [PBDR14b]. Die Idee hierbei ist es, dass Simulationen sowohl als eigenständige Komponenten betrachtet werden können, die spezifische

[3] `https://de.mathworks.com/products/matlab/`, aufgerufen am 1. Februar 2017
[4] `https://www.oracle.com/mysql/index.html`, aufgerufen am 1. Februar 2017

Fragestellungen beantworten und diese Funktionalität als Dienst bereitstellen, als auch Teil einer Co-Simulation sein können, bei der mehrere Simulationsdienste zur Beantwortung einer komplexen Fragestellung verbunden werden. Dadurch soll mit dem Konzept sowohl die Ausführung von dedizierten Mikro- oder Makrosimulationen als auch die flexible Komposition dieser ermöglicht werden. An diesem Punkt entscheidet sich der Ansatz aus [PBDR14b] von existierenden Ansätzen wie *HLA*, *GridLAB-D* oder *Mosaik* durch die serviceorientierte Sichtweise auf die einzelnen Simulationskomponenten. Im Rahmen der Komposition ist möglich, gängige Simulationswerkzeuge wie z.B. Multiagentenbasierte Simulationen (MABS) [Dav01] und MATLAB/Simulink Modelle als Sub- bzw. Co-Simulationen zu verbinden. So können hochskalierende Multiagentensysteme mit Standardtechnologien zur Modellierung von physikalischen Vorgängen gekoppelt werden.

Die entsprechende komponentenbasierte Architektur basiert auf dem Ansatz, dass alle Komponenten im System über einen *Service Bus* miteinander verbunden sind. Die Komponenten bieten über wohldefinierte Schnittstellen entsprechend Dienste an, die von anderen Komponenten aufgerufen werden. Simulationsszenarien werden in deklarativer Form mittels einer domänenspezifischen Sprache beschrieben und von einem *Scenario Manager* verarbeitet. Diese Beschreibungen beinhalten u.a. die Informationen, welche Simulationskomponenten für die Ausführung des Szenarios benötigt werden. Über den *Service Bus* wird eine entsprechende Anfrage vom *Scenario Manager* an den *Simulation Component Manager* geleitet. Dieses ist eine skalierbare Komponente, welche ggf. in mehrfacher Instanz und verteilt ausgeführt werden kann. Seine Aufgabe ist es, den Lebenszyklus der Simulationskomponenten zu handhaben. Er startet bei Bedarf weitere Komponenten eines Typs und leitet entsprechende Anfragen über den *Service Bus* an diese weiter. Die heterogenen Simulationskomponenten, die auf unterschiedlichen Technologien basieren können, werden über eine zugeordnete *Adaptive Manager* Komponente an das System angebunden. Dieser Manager maskiert die spezifischen Eigenschaften der Simulationskomponente über generische Schnittstellen (Black-box) und erweitert sie um adaptives Verhalten. Prinzipiell ähnelt der *Adaptive Manager* dabei im Aufbau und seiner Funktionalität dem *Automomic Manager* aus dem *Autonomic Computing* Ansatz (vgl. Abschnitt 2.2.3 bzw. [KC03]). Dadurch können auch bestehende Simulationssysteme integriert werden. Über entsprechende Subkomponenten kann der Ausführungszustand einer Simulationskomponente überwacht und ggf. die Ausführung gesteuert werden. Bei einer Simulationsanfrage kann der *Adaptive Manager* entweder auf bestehende Simulationsdaten (Datenbank) zurückgreifen oder eine entsprechende (Sub)-Simulation ausführen. Abbildung 5.2 stellt die entsprechende Servicekomponenten-basierte Architektur dar, die zur Realisierung von adaptiven Co-Simulationen verwendet werden kann.

Neben der Maskierung der spezifischen Eigenschaften der zugeordneten Simulationskomponente erfüllt der *Adaptive Manager* noch weitere, adaptive Aufgaben. Im Gegensatz zu Ansätzen wie *HLA*, *GridLAB-D* oder *Mosaik*

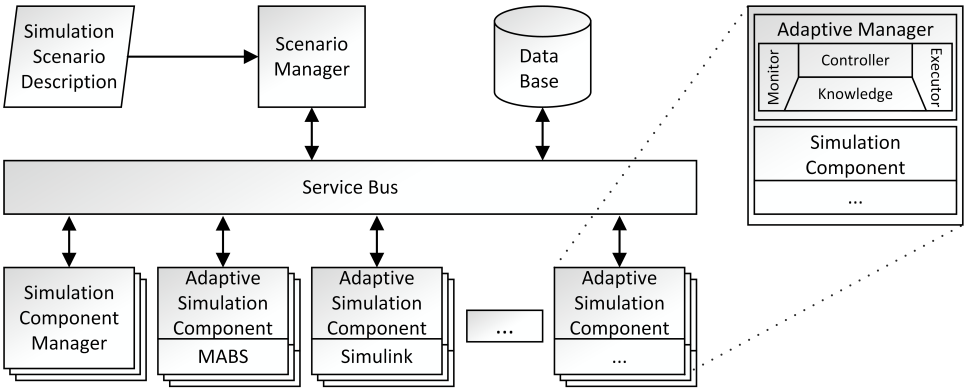



**Abbildung 5.2.:** Servicekomponenten-basierte Architektur für ein adaptives Co-Simulationssystem - aus (PBDR14b, S. 5).

befindet sich die Logik, die bestimmt, welche (Sub)-Simulationen für ein bestimmtes Simulationsszenario ausgeführt werden sollen, nicht in einer zentralen Manager-Komponente, sondern dezentral als Teil des *Adaptive Managers* in den einzelnen Simulationskomponenten. Dabei wird der *Adaptive Manager* selbstständig in der Lage sein, zu erkennen, wann eine Sub-Simulation ausgeführt werden muss. Dieser Ansatz basiert auf dem Grundgedanken, dass in dem System Simulationen von den Simulationskomponenten als Dienst angeboten werden und entsprechend von anderen Simulationen aufgerufen werden können, wenn diese für ihre Ausführung Simulationsergebnisse benötigen, die sie nicht selber erzeugen können. Durch diesen Ansatz wird es möglich sein, makroskopische Simulationen zu realisieren, die bei Bedarf auf Ergebnisse von mikroskopischen Simulationen zurückgreifen. Außerdem können komplexere, mesoskopische Simulationssysteme entworfen werden, die z.B. basierend auf einer Regelbasis entscheiden, in welchem Fall makroskopische Simulationen ausreichend sind und wann entsprechend mikroskopisch von einer anderen Komponente simuliert werden muss.

Damit die Simulationskomponenten in der Lage sind, die richtigen (Sub)-Simulationskomponenten zu finden, welche die benötigten Simulationen als Service anbieten, registrieren sich alle Simulationskomponenten bei dem *Simulation Component Manager* mit einer entsprechenden semantischen Beschreibung der Simulationsfunktionalität, die sie anbieten. Durch diese Architektur ist das System in der Lage, makroskopische Simulationen unter Berücksichtigung von mesoskopischen Effekten dynamisch durchzuführen und ermöglicht somit auch, Wechselwirkungen derselben zu untersuchen.

### 5.1.3. Skalierbare Integration von 4GL-Modellen

Mit der skalierbaren Integration von 4GL-Modellen in massive *Smart Grid* Simulationen und Applikationen wurde in [PBDR14a] eine Anwendung realisiert, welche die im vorangegangenen Abschnitt vorgestellte Architektur exemplarisch umsetzt. Unter der Abkürzung 4GL versteht man Program-

miersprachen der vierten Generation, dabei handelt es sich um spezielle Sprachen, die ihren Fokus auf *„Rapid Application Development“* [Mar91], also schnelle Anwendungsentwicklung legen. Der Ausdruck 4GL wurde dabei von *James Martin* in seinem Buch *„Application Development without Programmers“* [Mar00] geprägt. Programmiersprachen, die zu dieser Generation gehören, haben aktuell mit dem Aufkommen der modellbasierten Software-Entwicklung (siehe [BBG06]) wieder verstärkte Aufmerksamkeit erlangt. Beispiele für solche Sprachen sind u.a. *MATLAB*, *Simulink*[5], *Modelica* [FE98] und *GAMS*[6] (General Algebraic Modeling System). Diese Sprachen bzw. die Entwicklungswerkzeuge, zu denen sie gehören, werden in vielen Forschungsprojekten im Zuge des *„Rapid Prototyping“*, also der schnellen Erstellung von Prototypen, verwendet, um Modelle oder Algorithmen z.B. für Demand-Side-Management und die Integration von regenerativen Energiequellen in *Smart Grids* zu entwickeln (z.B. [BVR+13]). Diese Sprachen reduzieren den Entwicklungsaufwand durch die Verwendung von leicht verständlichen, anwendungsorientierten Paradigmen. Weiterhin bieten sie eine gute Lesbarkeit, was den Quellcode angeht, und die benutzerorientierte Repräsentation von 4GL Programmen erleichtert deren Wartbarkeit und Erweiterbarkeit.

Allerdings ist die Integration solcher 4GL Modelle oder Algorithmen in operative Systeme oder auch hoch-skalierende Simulationen eine erhebliche Herausforderung, da diese i.d.R. in einer speziellen Laufzeitumgebung ausgeführt werden müssen. Selbst wenn es möglich ist, die entsprechenden Modelle so zu exportieren, dass sie als unabhängige, externe Modelle in die Infrastruktur des operativen Systems eingebunden werden können, bleibt das Skalierungsproblem bestehen. Eine mögliche Lösung hierbei ist es, die entsprechenden Algorithmen und Modelle neu zu schreiben, sodass sie die gleiche Programmiersprache wie das entsprechende Operativsystem bzw. die Simulation nutzen. Dieses löst zwar Integrations- und Skalierungsprobleme, sorgt allerdings für neue Probleme und Aufwände, da die entsprechend neugeschriebenen Komponenten den kompletten Prozess der Validierung und Verifizierung erneut durchlaufen müssen, um ihr korrektes Verhalten sicherzustellen. Zusätzlich kann die Neuimplementierung der Algorithmen und Modelle als zeitaufwändig angesehen werden, da die neuen Zielprogrammiersprachen i.d.R. weniger anwendungsspezifische Sprachelemente mitbringen.

Daher wird im Folgenden ein skalierbares Integrationskonzept für 4GL-Modelle und Algorithmen beschrieben, das auf Standard Web-Technologie basiert und die im vorangegangenen Abschnitt beschrieben Architektur exemplarisch implementiert.

### Integrationskonzept

Zur skalierbaren Integration von 4GL-Modellen werden die Funktionen als *REST* [Fie00] Web-Services im Sinne einer Serviceorientierten Architek-

[5] `https://de.mathworks.com/products/simulink/`, aufgerufen am 1. Februar 2017
[6] `https://www.gams.com/`, aufgerufen am 1. Februar 2017

tur (SOA) [Erl04] gekapselt. Dadurch ist jedes 4GL Modell, das in einem Produktiv- oder Simulationssystem aufgerufen werden soll, als *REST* Servicemethode zugreifbar. In [PBDR14a] wurde die exemplarische Integration von Programmcode, der aus MATLAB mit dem *Builder JA*[7] exportiert wurde, untersucht. Der *Builder JA* ist ein Programm, das es erlaubt, MATLAB Quellcode als Java Archive (JAR) zu exportieren, sodass der Code direkt aus einer Java-Anwendung heraus aufgerufen werden kann. Allerdings muss dazu auf dem Rechner, auf dem der MATLAB Code aus Java heraus aufgerufen werden soll, die *MATLAB Compiler Runtime* (MCR) installiert sein. Bei der MCR handelt es sich um eine Sammlung von dynamischen Bibliotheken, die von MATLAB Anwendungen aufgerufen werden können, welche außerhalb einer MATLAB Installation ausgeführt werden sollen. Aufgrund der Threadsicherheit der MCR ist es allerdings nicht möglich, auf diese parallel aus einem ausführenden Prozess zuzugreifen. Daher werden mehrere Ausführungsprozesse benötigt, wenn eine skalierbare, parallele Ausführung erreicht werden soll.

Abbildung 5.3 zeigt die angedachte Lösung: Die Funktionen, die von den erstellten JAR Dateien angeboten werden, werden als REST Webservices gekapselt und auf einen *Apache Tomcat*[8] Anwendungsserver veröffentlicht. Jede *Tomcat* Instanz ist ein eigenständiger Ausführungsprozess, sodass die parallele Ausführung der exportieren MATLAB Funktionen ermöglicht wird. Daher werden mehrere Instanzen benötigt, um eine skalierbare Lösung zu ermöglichen. Der in [PBDR14a] vorgestellte Ansatz, verwendet dazu einen Lastverteiler, der die *REST* Webservice Aufrufe auf mehrere *Tomcat* Instanzen verteilt. Dazu kann z.B. ein *Apache HTTPD*[9] Webserver in Verbindung mit der *mod_jk*[10] Erweiterung verwendet werden. Diese Konfiguration benutzt einen *Round Robin* Mechanismus zur Lastverteilung. Der Lastverteilungsmechanismus wird dabei transparent für die aufrufende Anwendung zur Verfügung gestellt. Im Fall von Performanz Problemen können einfach weitere *Tomcat* Instanzen hinzugefügt werden, ohne das Änderungen auf der aufrufenden Seite vorgenommen werden muss. Außerdem ist es möglich, den Lastverteilungsalgorithmus auszutauschen, sodass komplexere oder anwendungsabhängige Lösungen verwendet werden können.

Das skalierbare Integrationskonzept von 4GL-Komponenten kann auch auf den Kontext der Servicekomponenten-basierten Architektur für Simulationssysteme aus den vorherigen Abschnitt (siehe Abbildung 5.2) abgebildet werden. Basierend auf den Konzepten, die in dem Abschnitt vorgestellt wurden, kann der *Apache HTTPD* Server, der als Lastverteiler verwendet wird, die Rolle des *Adaptive Manager* übernehmen. Dieser verarbeitet die Serviceanfragen, überwacht die Ausführung und den Zustand der *Tomcat* Instanzen und

[7] `http://www.mathworks.de/products/javabuilder/index.html`, aufgerufen am 1. Februar 2017

[8] `http://tomcat.apache.org/`, aufgerufen am 1. Februar 2017

[9] `https://httpd.apache.org/`, aufgerufen am 1. Februar 2017

[10] `http://tomcat.apache.org/connectors-doc/`, aufgerufen am 1. Februar 2017

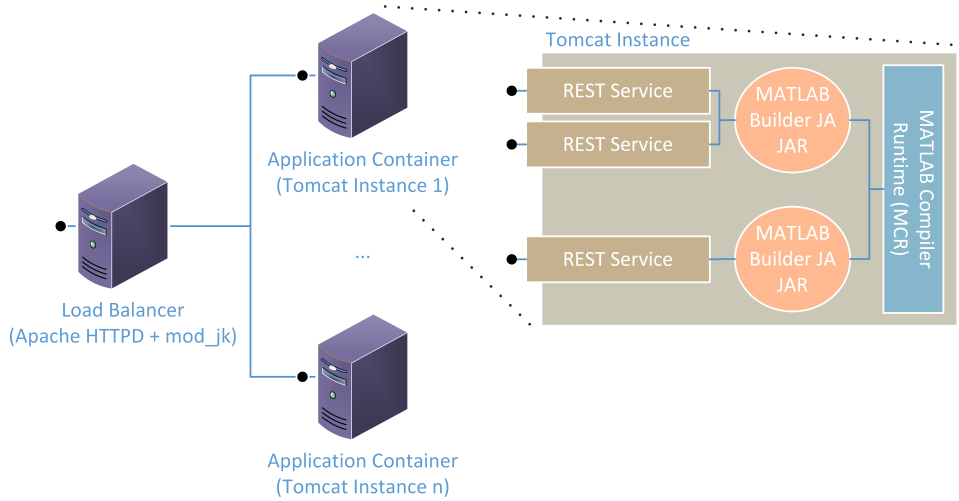



**Abbildung 5.3.:** Skalierbare Integration von MATLAB Code - aus (PBDR14a, S. 5).

ruft die entsprechenden Webservices auf, die von dem *Tomcat* Container angeboten werden. In diesem Kontext wird eine *Tomcat* Instanz, die ein 4GL-Modul kapselt, als Simulationskomponente angesehen, die einen bestimmten Simulationsdienst anbietet. Entsprechend zu z.B. einem Multiagentensystem, das aus mehreren Agenten besteht, kann so eine skalierbare 4GL-Modell Simulationskomponente zwecks Skalierung und Lastverteilung aus mehreren *Tomcat* Instanzen bestehen. Abbildung 5.4 illustriert diesen Sachverhalt.

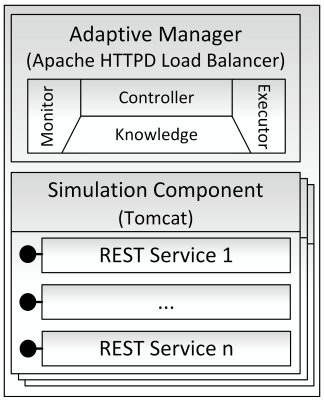



**Abbildung 5.4.:** Integration einer skalierbaren 4GL-Komponente in die Architektur für ein servicebasiertes Simulationsframework (siehe Abbildung 5.2).

### Kombinierte Hard- und Software Smart Grid Simulationen

Da die standardisierten REST Webservices Schnittstellen in unterschiedlichen weiträumigen Netzwerken (WAN), wie z.B. dem Internet, verwendet werden können, kann die vorgestellte Lösung zur Integration von 4GL-Modellen auch dazu genutzt werden, um kombinierte Hard- und Software Simulationen zu realisieren. Solche *cyber-physischen* Systeme bezeichnen einen Verbund aus in-

formatischen, softwaretechnischen Komponenten mit mechanischen und elektronischen Teilen, die über eine Dateninfrastruktur, z.B. das Internet, kommunizieren. Gerade im Bereich der *Smart Grids* gewinnen solche Systeme zunehmend an Bedeutung (siehe [Kar11]). Eine mögliche Lösung hierbei ist, dass die physischen Geräte ebenfalls über REST Schnittstellen angesprochen werden, um so mit anderen Softwarekomponenten zu kommunizieren.

Abbildung 5.5 illustriert dieses Konzept. Das dargestellte *Smart Grid* Simulationsmodell besteht aus drei unterschiedlichen Domänen. Die Benutzerdomäne (*User Domain*) bildet die Benutzerinteraktionen mit dem System ab. Dabei kann es sich um einen technischen Benutzer handeln, der die Simulationen startet und stoppt oder um einen funktionalen Benutzer, der das Simulationssystem verwendet, um ein Systemverhalten zu validieren. Die Simulations- und Kontrolldomäne (*Simulation and Control*) bildet die simulierten Einheiten ab. Das können z.B. Modelle von Haushalten oder Geräten sein, die ihre physischen Gegenstücke simulieren oder für die Kontrolle bzw. Regulation der Strukturen im System zuständig sind. In dieser Domäne könnte der vorgestellte Ansatz zur skalierbaren Integration von 4GL-Modellen seine Anwendung finden. Die letzte Domäne ist die physische (*Physical*), hier befinden sich die tatsächlichen Geräte. Ähnlich wie in einer *Hardware-in-the-Loop* Simulation (siehe [ISS99]) können die simulierten Entitäten entweder verwendet werden, um die physischen Geräte zu testen oder die physischen Geräte können verwendet werden, um die softwaretechnischen Kontrollstrukturen und -algorithmen in Verbindung mit simulierten Geräten zu testen.

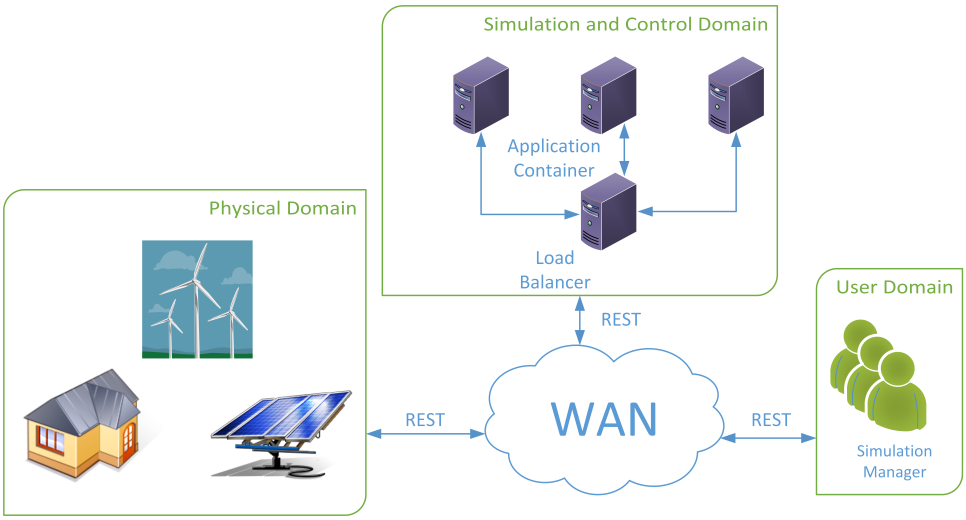



**Abbildung 5.5.:** Kombiniertes Hard- und Software Smart Grid Simulationsmodell - aus (PBDR14a, S. 7).

## 5.2. Simulation as a Service

Ein spezifischer Ansatz, der sich mit der Interoperabilität von Simulationskomponenten und deren skalierbarer Ausführung beschäftigt, ist der *Simulation as a Service* Ansatz. Bei diesem Ansatz werden einzelne Simulationskomponenten oder ganze Simulationen als Dienste in einer *Cloud Computing* [Sos11] Infrastruktur angeboten. Im Folgenden wird zunächst ein Überblick über *Cloud Computing* gegeben und es werden aktuelle Technologien vorgestellt, bevor auf existierende Ansätze, die Simulationen als Dienste anbieten, eingegangen wird. Schließlich wird ein spezifischer Ansatz zur Umsetzung dieses Konzept, der im Rahmen dieser Dissertation in [PDR15c] erarbeitet wurde, vorgestellt. Basierend auf diesem Ansatz wird auf grundsätzliche Herausforderungen und Alternativen eingegangen, die bei der Umsetzung dieses Konzepts entstehen. Abschließend wird ein (Daten-)adaptiver Ansatz vorgestellt, der in [PDR15a] erarbeitet wurde und ein Beispiel für eine adaptive Simulation, die als ein Service angeboten wird, bietet.

### 5.2.1. Cloud Computing Technologien

*Cloud Computing* kann als neuartiger Ansatz zum Management von IT-Infrastrukturen verstanden werden. Dabei werden Programme nicht mehr lokal auf dem Rechner, auf dem sie installiert sind, ausgeführt, sondern auf einem anderen Rechner, der aus der Ferne, z.B. über das Internet, aufgerufen wird. Technischer formuliert ist Cloud Computing ein Ansatz, bei dem IT-Infrastrukturen (z.B. Rechenkapazität, Datenspeicher oder Netzkapazitäten) über ein Netz zur Verfügung gestellt werden. Angebot und Nutzung dieser Dienstleisten erfolgen dabei über technische Schnittstellen und entsprechende Protokolle. Die Autoren von [BPJ11] rücken dabei das Argument in den Vordergrund, dass Ideen aus dem Cloud Computing zur optimalen Verwendung von Ressourcen auch verwendet werden können, um die freien Ressourcen existierender Computer in einem Firmen- oder Hochschulnetzwerk zur Verfügung zu stellen. Das amerikanische *National Institute of Standards and Technology* (NIST) veröffentlichte 2009 eine Definition [MG09], die weitgehend auf Akzeptanz stieß und drei unterschiedliche Servicemodelle definiert:

**Infrastructure as a Service (IaaS):** Hierbei bieten Cloud-Infrastrukturen Nutzungszugang zu virtualisierten Computerhardware-Ressourcen wie Rechnern, Netzen oder Speicher an. Mit IaaS gestalten die Benutzer frei ihre eigenen virtuellen Computer-Cluster und sind selber für die Auswahl, die Installation und den Betrieb ihrer Software verantwortlich.

**Platform as a Service (PaaS):** Hier bieten Rechnerwolken Nutzungszugang zu Programmier- und Laufzeitumgebungen mit flexibel und dynamisch anpassbaren Rechen- und Datenkapazitäten. Mit PaaS entwickeln Nutzer ihre eigenen Anwendungen oder lassen diese innerhalb der Softwareumgebung, die vom Dienstanbieter bereitgestellt und unterhalten wird, ausführen.

**Software as a Service (SaaS):** Rechnerwolken bieten hier direkt Nutzungszugang zu Anwendungsprogrammen. SaaS Dienstanbieter bieten eine spezielle Auswahl von Software an, die auf ihrer Infrastruktur läuft. SaaS wird daher auch als *software on demand*, also Software bei Bedarf bezeichnet. Die Anwendungen, die hier als Dienste angeboten werden, nutzen dabei i.d.R. IaaS oder PaaS Infrastrukturen.

Grundsätzlich gibt es viele verschiedene Ansätze und Technologien für die Implementation und den Betrieb von Cloud-Anwendungen. Im Rahmen dieses Abschnitts werden exemplarisch zwei unterschiedliche PaaS Lösungen vorgestellt. PaaS Lösungen eignen sich besser für die Realisierung von Simulationen, die in Form eines Services angeboten werden, als IaaS Lösungen, da sie bereits vom Betriebssystem und der Hardware abstrahieren und spezielle Schnittstellen für die Anwendungsprogrammierung (API) bieten, die es ermöglichen, skalierbare Anwendungen auf Basis der Cloud Infrastruktur zu realisieren.

Die *Mesos* Plattform [HKZ+11] bietet eine Abstraktionsschicht zur gemeinsamen Nutzung von Ressourcen für Cloud-Anwendungen. Dabei wird eine Schnittstelle zur Verfügung gestellt, die es Anwendungen ermöglicht, feingranular auf Ressourcen in einem Rechen-Cluster zuzugreifen und diese zu teilen. Der Fokus von *Mesos* liegt dabei auf der Unterstützung von unterschiedlichen Rahmenwerken, die ihrerseits wieder die Implementierung von skalierbaren, verteilten Anwendungen ermöglichen, z.B. *Apache Hadoop* [ABPA+09] oder *MPI* (Message Passing Interface) [GLS99]. Dadurch, dass diesen unterschiedlichen Rahmenwerken eine gemeinsame Schnittstelle zur Verfügung gestellt wird, wird die Ausnutzung der Ressourcen in einem Rechen-Cluster verbessert, indem die parallele Ausführung unterschiedlicher Rahmenwerke unterstützt wird. Ressourcen werden in *Mesos* dabei über einen verteilten 2-Level Scheduling-Mechanismus, genannt *resource offers*, geteilt. Ein solches *Ressourcen-Angebot* kapselt eine Reihe von unterschiedlichen Ressourcen, die vom einem Rahmenwerk allokiert werden können, um seine Tasks auszuführen. *Mesos* entscheidet dabei darüber, welche Ressourcen in welcher Zahl welchem Rahmenwerk zur Verfügung gestellt werden und die Rahmenwerke entscheiden darüber, welche Angebote sie annehmen, um welche Berechnungen auszuführen. Damit gibt *Mesos* die Kontrolle über das weitere Scheduling an die Rahmenwerke ab. Dieses dezentrale Scheduling-Model findet möglicherweise nicht immer eine global optimale Lösung, aber nach [HKZ+11] führt es in der Praxis zu ausreichend guten Ergebnissen.

Während der Fokus von *Mesos* auf dem Teilen von Ressourcen in Rechen- und Datenzentren liegt, bietet *JadexCloud* [BPJ11] eine PaaS Infrastruktur, um verteilte Anwendungen zu entwickeln und bereitzustellen, wobei der Fokus auf der Verteilungstransparenz liegt. *JadexCloud* ist dabei eine Weiterentwicklung der *Jadex* Multiagentenplattform [BP12c] und ermöglicht die Entwicklung von cloud-basierten Agenten-Anwendungen mit Service-orientierter Kommunikation. Das Konzept basiert dabei auf einem 3-Schichten Modell, das

hilft, die Verantwortlichkeiten zu separieren und die Komplexität von Anwendungen zu managen. Der sogenannte *daemon layer* stellt dabei eine Infrastruktur zum Verwalten von Cloud-Ressourcen bereit. Dabei werden Knoten (Rechner), die dem Rechenverbund beitreten und diesen verlassen, automatisch detektiert, sodass die Struktur des Systems entsprechend automatisch adaptiert werden kann. Die *platform layer*, die auf der *daemon layer* aufbaut, kümmert sich um Management-Aufgaben, die mit den Anwendungen zu tun haben. Dabei werden Anwendungsartefakte automatisch auf unterschiedliche Knoten veröffentlicht und bei Bedarf die entsprechenden Anwendungskomponenten gestartet oder gestoppt. Schließlich stellt die *application layer* eine Programmierschnittstelle zur Realisierung eigener Anwendungen auf Basis der *JadexCloud*-Infrastruktur zur Verfügung und unterstützt die Entwicklung von verteilten Anwendungen mit speziellen Werkzeugen zum Debuggen. In der aktuellen Version der bereitgestellten PaaS-Infrastruktur [BJP14] wird außerdem die Verwaltung von nicht-funktionalen Anforderungen unterstützt. Damit ist es möglich, nicht-funktionale Eigenschaften für Dienste zu definieren, die automatisch überwacht werden. Ein Beispiel für eine solche nicht-funktionale Eigenschaft ist eine Warteschlangen-Eigenschaft, die zählt, wie viele Anfragen momentan auf Beantwortung warten. Basierend auf diesen nicht-funktionalen Eigenschaften ist es möglich, die Dienstauswahl zu automatisieren, indem entsprechende nicht-funktionale Anforderungen definiert werden wie z.B. den Dienst mit der kleinsten Warteschlangenlänge zu finden.

### 5.2.2. Existierende Ansätze

Dass die Bereitstellungen von Simulationen als Dienste noch ein recht junger Ansatz ist zeigt sich u.a. darin, dass noch keine einheitliche Terminologie gefunden wurde. Allerdings besteht bereits eine inhaltliche Nähe der Begrifflichkeiten. Die Autoren von [GBH11] sprechen in diesem Zusammenhang von Simulationssoftware, die im Rahmen einer Service-orientierten Architektur (SOA) als Dienste angeboten werden (*SimSaaS*). Der begriffliche Fokus liegt hierbei also auf einer Spezialisierung der bereits üblichen *Software as a Service* (SaaS) Angebote im Cloud Computing Umfeld. Im Rahmen der Arbeit von [Cay13] wird der Begriff der „Modellierung und Simulation als ein Service“[11] für angebotene Modellierungs- und Simulationsdienste eingeführt (*MSaaS*). Die Modellierung, die der eigentlichen Simulation vorangeht, ist hierbei also explizit Teil der angebotenen Dienste. Ein ähnlicher Ansatz wird von den Autoren von [BCB+14] verfolgt, wobei hier, zumindest auf begrifflicher Ebene, den Simulationen im „Simulation und Modellierung als ein Service“ Ansatz[12] (*SMaaS*) eine wichtigere Bedeutung als den Modellierungsaspekten zugeordnet wird.

Dabei beschreiben die Autoren von [GBH11] mit dem *SimSaaS* einen Ansatz, bei dem Simulationsexperimente in der Form modelliert werden, dass mehrere Simulationsdienste so verbunden werden, das daraus ein Workflow

[11] *„Modeling and Simulation as a Service“* im Englischsprachigen Original.
[12] *„Simulation and modeling as a service“* im Englischsprachigen Original.

entsteht, der vorgibt, wie das Experiment ausgeführt werden soll. Die Autoren unterscheiden Simulationsservices dabei von normalen Web Services in der Form, dass die Abbildung von zeitlichen Vorgängen und der Zustand der Simulation essentiell sind. Daher werden die Simulationsservices als zustandsbehaftet betrachtet, die jede Anfrage als eine Serie von abhängigen Transaktionen betrachten, die in einer Beziehung zu vorherigen Anfragen stehen und abhängig vom aktuellen Zustand des Modells sind. Daher werden im Rahmen von [GBH11] unterschiedliche Kommunikations- und Synchronisationsparadigmen beschrieben und im Bezug auf ihre Eignung zur Verbindung von zustandsbehafteten Simulationsservices bewertet.

Im Rahmen von [Cay13] wurden aktuelle Trends und Technologien im Umfeld von Modellierungs- und Simulationsdiensten als Cloud-Services untersucht. Dabei wird „Modellierung- und Simulation als ein Service“ als ein Modell definiert, bei dem Modellierungs- und Simulationsdienste von einem *Cloud Service Provider* (Dienstanbieter) angeboten werden. Die zugrundeliegende Infrastruktur, die Plattform und die Software-Anforderungen sind dabei für den Benutzer unsichtbar. In diesem Fall ist der Dienstanbieter verantwortlich für die benötigten Lizenzen, Software-Updates, die Skalierung der Infrastruktur und allgemein für die Qualitätsanforderungen, die meistens in Form von *Service Level Agreements* (SLA) definiert sind. Das resultierende Modell bietet also Zugriff auf die in einer Cloud-Umgebung bereitgestellten Anwendungen und erlaubt es den Benutzern, diese Dienste zu geringen Kosten und im Rahmen ihrer jeweiligen Skalierungsanforderungen flexibel zu nutzen. Ein sehr ähnliches Modell wurde in der Arbeit von [BCB+14] mit dem *Polymer Portal* vorgestellt. Auch hier geht es darum, Modellierungs- und Simulationsdienste im Rahmen einer skalierbaren Cloud-Umgebung Endnutzern zur Verfügung zu stellen.

Das Bereitstellen von Modellierungs- und Simulationsdiensten findet u.a. auch Anwendung im Bereich der Multiagentenbasierten Simulationen, so bietet z.B. das *MARS* Multiagenten-Framework [HWFTC14] eine Möglichkeit, um Modellierungs- und Simulationsdienste einem Endnutzer anzubieten [HATC+16]. Dabei wird es Anwendungsexperten ermöglicht, ihre Daten und Modelle im Rahmen einer Web-Oberfläche zu integrieren. Die Komplexität der skalierbaren Ausführung der verteilten Simulation wird dabei, für den Anwender transparent, im Hintergrund durch einen sogenannten *Agent Shadowing* Mechanismus übernommen. Durch eine Schicht-bezogene Segmentierung der Modelle soll es den Anwendern und Entwicklern ermöglicht werden, sich auf einen spezifischen Aspekt zur Zeit zu konzentrieren, während sie ihre Simulationsmodelle entwickeln.

### 5.2.3. Konzept und Architektur

Der *Simulation as a Service* Ansatz kann in Form einer konzeptionellen Architektur umgesetzt werden, indem das *SaaS* Konzept aus dem Cloud Computing Umfeld adaptiert wird, um Simulationen als Dienste in einer Cloud-

Infrastruktur bereitzustellen. Die Idee dabei ist, dass Simulationen innerhalb sogenannter *Simulation Service Components* gekapselt werden. Diese Komponenten kapseln die Funktionalität der Simulation und stellen eine Service-Schnittstelle zur Verfügung, über die die Simulationen aufgerufen werden können. Durch den Ansatz wird dabei der Aufruf von unterschiedlichen Simulationstypen vereinheitlicht, während gleichzeitig die Kombination von unterschiedlichen Implementierungsmodellen und Programmiersprachen unterstützt wird. In Abbildung 5.6 werden dabei beispielhaft ein 4GL-Modell einer Haushaltskomponente und eine komplette Multiagenten-Simulation innerhalb einer *Simulation Service Component* gekapselt. Im Falle der Bereitstellung und Einbindung einer multiagentenbasierten Simulation sind grundsätzlich zwei Arten der Einbindung denkbar. Betrachtet man eine relativ einfache Multiagenten-Simulation, die auf einem einzelnen Knoten im Netzwerk ausgeführt wird, so kann die Simulation als Ganzes über eine *Simulation Service Component* bereitgestellt werden. Dadurch wird ein vereinheitlichter service-orientierter Aufruf ermöglicht. Denkbar ist aber ebenfalls der komplexere Fall, bei dem eine verteilte Multiagenten-Simulation betrachtet wird. In diesem Fall wäre ein möglicher Integrationsansatz die Entwicklung einer dedizierten Starter Komponente, welche die verteilte Anwendung startet und die Ergebnisse einsammelt, um diese über die zu bereitstellende Schnittstelle zurückzugeben.

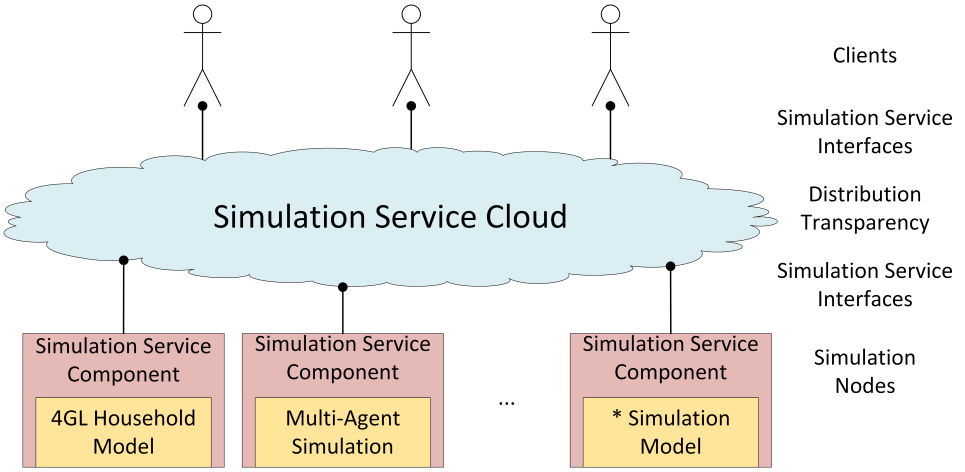



**Abbildung 5.6.:** Konzeptionelle Architektur für den Simulation als ein Service Ansatz - aus (PDR15c, S. 4).

Grundsätzlich können Simulationen, die als Services angeboten werden, entweder direkt von einem Client aufgerufen werden, wenn dieser eine bestimmte Simulationsfunktionalität benötigt, oder es liegt der Fall einer komplexen Simulation vor, die wiederum aus mehreren Simulationsdiensten zusammengesetzt ist. In einem solchen Fall kann eine Simulationskomponente die Dienste, die von anderen Simulationskomponenten angeboten werden, nutzen, falls sie Ergebnisse dieser anderen Simulationen für die eigene Berechnung benötigt. Ein Beispiel für einen solchen Anwendungsfall ist die Realisierung

einer mesoskopischen Co-Simulationen, bei der eine makroskopische Simulation die Ergebnisse von mikroskopischen Teil-Simulationen benötigt. Zur Komposition von unterschiedlichen Simulationsdiensten können Ansätze wie *Web Service Business Process Execution Language* (WS-BPEL) [OAS07] oder *Business Process Model and Notation* (BMPN) [OMG11] zur Orchestrierung von Web-Services für die Simulationsdomäne adaptiert werden.

Im Wesentlichen gibt es zwei Paradigmen für die Komposition und Ausführung von Simulationsdiensten: Hierarchisch und parallel. Im Fall einer hierarchischen Struktur sind die Teil-Simulationen nur aktiv, wenn sie von der Haupt-Simulation zur Bereitstellung von Ergebnissen aufgerufen werden. Spielt Zeit im Rahmen dieser Simulationen eine Rolle, so kann diese entweder von der Haupt-Simulation verwaltet und als Parameter bei Aufruf an die Teil-Simulationen weitergeleitet werden, oder jede Teil-Simulation verfügt über ihr eigenes Zeit-Modell und verwaltet dieses eigenständig. In diesem Fall ist es möglich, Simulationen mit unterschiedlichen Zeitskalen zu kombinieren. Abbildung 5.7 zeigt die hierarchische Komposition und Ausführung von Simulationen in Form eines UML-Sequenzdiagramms. Gezeigt wird eine Simulation $A$, die während ihrer Ausführung die Ergebnisse der Teil-Simulationen $B$ und $C$ benötigt. Die Teil-Simulation $C$ benötigt dabei wiederum Ergebnisse der Teil-Simulation $D$, um die Anfrage von Simulation $A$ zu beantworten.

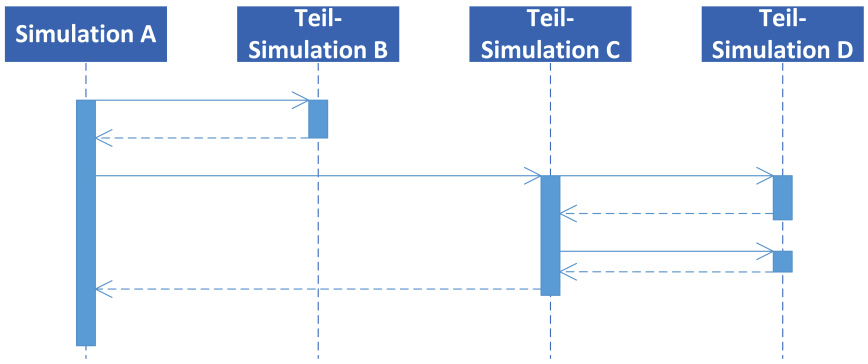



**Abbildung 5.7.:** UML Sequenzdiagramm das die hierarchische Komposition und Ausführung von Simulationsdiensten zeigt - nach (PDR15c).

Im Fall einer der parallelen Komposition werden Simulationen parallel ausgeführt und tauschen während ihrer Ausführung Daten aus. Dieses Paradigma entspricht dem *SimSaaS* Ansatz, der in [GBH11] beschrieben wurde. Die Autoren von [GBH11] unterscheiden dabei Simulationsdienste von klassischen Web-Services dadurch, dass Erstere als zustandsbehaftete Dienste betrachtet werden müssen, bei denen jede Anfrage als Teil einer Serie von abhängigen Transaktionen, die im Bezug zu den vorherigen Anfragen und dem aktuellen Zustand des Modells stehen, behandelt wird. Ein entscheidender Aspekt bei diesem Kompositionsparadigma der verketteten Simulationen ist es, sicher zu stellen, dass das zeitliche Verhalten der Simulation synchronisiert ist, so-

dass Ereignisse in der korrekten Reihenfolge verarbeitet werden. Das parallele Kompositionsparadigma ist in Abbildung 5.8 dargestellt.

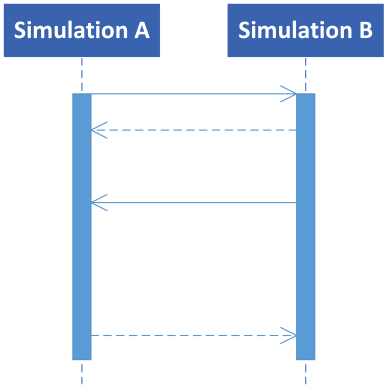



**Abbildung 5.8.:** UML Sequenzdiagramm das die parallele Komposition und Ausführung von Simulationsdiensten zeigt - aus (PDR15c, S. 5).

### 5.2.4. Alternativen bei der Implementierung

Zur Implementierung des *Simulation as a Service* Ansatz stehen unterschiedliche Alternativen zur Verfügung. Grundsätzlich ist es dabei sinnvoll, etablierte Technologien und Ansätze aus dem Umfeld der Service-orientierten Architekturen zu verwenden (vgl. z.B. [PvdH07]). Werden die Anforderungen um Verteilungstransparenz, Skalierbarkeit und Robustheit ergänzt, ist es sinnvoll, die gewählten Technologien um etablierte Technologien aus dem *Cloud Computing* Umfeld (siehe Abschnitt 5.2.1) zu ergänzen. Wenn die Simulation, die dabei als Dienst bereitgestellt werden soll, als *Black-box* betrachtet wird, die ohne ein explizites Konzept zur Verteilung innerhalb einer *Cloud Computing* Infrastruktur entwickelt wurde, können die gewünschten nichtfunktionalen Anforderungen durch die Verwendung des IaaS Ansatzes (vgl. Abschnitt 5.2.1), bei dem von der physikalischen Hardware abstrahiert wird, erreicht werden (vgl. [PDR15c]). In diesem Fall ist der IaaS Ansatz besser geeignet als der PaaS Ansatz, da Letzterer bedingt, dass die entsprechende Anwendung bzw. Simulation explizit gegen die Schnittstelle der Cloud-Plattform entwickelt wurde. Natürlich ist es auch möglich, eine *Black-box* Simulation durch eine Komponente zu kapseln, die entsprechend gegen die Schnittstelle der PaaS Plattform programmiert wurde, allerdings wird die Simulationskomponente in diesem Fall nicht von den Verteilungs-, Skalierbarkeits und Robustheits-Eigenschaften der Plattform profitieren. Daher sollten bestehende Simulationssysteme, die als Service innerhalb einer Cloud-Umgebung bereitgestellt werden sollen, auf Basis des IaaS Ansatzes realisiert werden. Für den Fall, dass eine Simulation vom Grund auf neu entwickelt werden soll oder das Simulationsmodell nur über einen geringen Ressourcenbedarf verfügt, ist es von Vorteil, die Simulation auf Basis des PaaS Ansatzes zu realisieren, um direkten Einfluss auf die Verteilungs- und Skalierbarkeitskonzepte nehmen zu können (vgl. [PDR15c]).

Im Rahmen von [PDR15c] wurden exemplarisch drei Implementierungsalternativen vorgestellt und im Hinblick auf ihre Skalierbarkeit evaluiert. Dabei wurde ein 4GL-Modell, das den Energieverbrauch eines mit einer elektrischen Heizung ausgestatteten Haushalts simuliert, als Dienst bereitgestellt. Die erste Implementierungsalternative folgte dabei dem Ansatz zur skalierbaren Integration von 4GL-Modellen, der bereits in Abschnitt 5.1.3 vorgestellt wurde. Hier wurde das Simulationsmodell des Haushalts durch einen REST Web-Service gekapselt und die Skalierbarkeit durch einen Lastverteiler auf Basis des *Apache HTTP* Servers in Kombination mit dem *mod_jk* Tomcat Connector realisiert. Die Lastverteilung verwendete dabei ein *Round Robin*-Verfahren zum Scheduling.

Die zweite Implementierung verwendete den Ansatz zur Realisierung von elastischen Komponenten-basierten Cloud-Anwendungen, der in [BJP14] vorgestellt wurde. Bei diesem Ansatz wird auf Basis der *Jadex*-Plattform das Konzept der Serviceauswahl anhand von nicht-funktionalen Eigenschaften verwendet. Das 4GL Haushaltsmodell wird auch in diesem Fall durch eine entsprechende Service-Komponente gekapselt, wobei hierbei jeder Service mit einem Observer für nicht-funktionale Eigenschaften ausgestattet wird, der die Länge der Warteschlange für den Service observiert. Ein Client, der den Simulationsservice anfragen will, kann nun eine Evaluationskomponente verwenden, welche die Länge der Warteschlangen aller verfügbaren Services evaluiert und den Service mit der kleinsten Warteschlange für die Anfrage auswählt. Somit kann eine optimale Verteilung der Service-Anfragen erzielt werden.

Die dritte Implementierungsalternative verwendete die *JadexCloud* PaaS-Plattform [BPJ11]. Wie andere PaaS-Lösungen auch, vereinfacht *JadexCloud* die Konstruktion von Cloud-Anwendungen durch die Bereitstellung von entsprechenden Schnittstellen. Im Bezug auf die Entwicklung von verteilten Anwendungen bietet *JadexCloud* zwei grundlegende Vorteile: Die Bereitstellung von Diensten, die von Service-Komponenten angeboten werden, in einem von der Plattform kontrollierten Service-Pool und die dynamische Verteilung von Service-Komponenten auf neue Jadex-Knoten im Netzwerk. Zunächst kann jeder Dienst, der von einer Komponente angeboten wird, in einem von der Plattform kontrollierten Service-Pool bereitgestellt werden. Durch die Verwendung des Service-Pools müssen sich Entwickler nicht mehr selbst um das Starten und Beenden der entsprechenden Service-Komponenten kümmern. Der Service-Pool übernimmt diese Aufgaben und startet automatisiert neue Service-Komponenten, sobald neue Ressourcen benötigt werden, um entsprechende Anfragen zu beantworten. Daher agiert der Service-Pool als ein Proxy, der alle ankommenden Anfragen an Service-Komponenten mit verfügbaren Ressourcen weiterleitet. Die Verwendung des Service-Pools unterstützt dabei nicht nur die Realisierung von skalierbaren Anwendungen, sondern unterstützt ferner die dynamische Verteilung der Service-Komponenten auf neue Jadex-Knoten im Netzwerk. Dabei wird automatisch detektiert, wenn neue *JadexCloud*-Plattformen im Netzwerk gestartet werden, sodass bei Bedarf neue Instanzen der Service-Komponenten auf den entfernten Plattformen

veröffentlicht werden können, sollten die bestehenden Instanzen die ankommenden Anfragen nicht mehr alleine beantworten können.

In Abbildung 5.9 werden Konfigurationsbeispiele für die drei Implementierungsalternativen dargestellt. Diese wurden im Rahmen der Analyse in [PDR15c] verwendet, um die Implementierungsalternativen auf ihre Skalierbarkeit hin zu untersuchen. Dabei stellt in jedem der drei Fälle ein Client entsprechende Anfragen an die Haushaltskomponente, deren Energieverbrauch im Rahmen einer Simulation bestimmt wird:

- Bei der Implementierung, die auf REST Web-Services basiert, handelt es sich dabei um einen HTTP Client, der die Anfrage über das REST Protokoll stellt. Ein *Apache HTTP* Server übernimmt die Aufgabe des Lastverteilers, während das Simulationsmodell als REST Web-Service bereitgestellt wird, wofür ein *Apache Tomcat* Server als Anwendungscontainer verwendet wird.

- Im Fall der Implementierung, die auf der Serviceauswahl basierend auf der nicht-funktionalen Eigenschaft der Warteschlangenlänge beruht, ist ein Client für die entsprechende *Jadex*-Plattform realisiert. Das Simulationsmodell wird als *Jadex*-Service gekapselt und dementsprechend auf einer *Jadex*-Plattform ausgeführt. Die Anfragen des Clients richten sich dabei zunächst an die Evaluationskomponente, welche die nicht-funktionalen Eigenschaften der Services abfragt und den Besten zurückgibt. Der Client fragt dann in einem zweiten Schritt direkt diesen Service an.

- Für die Implementierung der *JadexCloud* Lösung ist der Client erneut für die *Jadex*-Plattform realisiert. Wieder ist das Simulationsmodell als *Jadex*-Service gekapselt und wird innerhalb der *Jadex*-Plattform ausgeführt. Die Anfragen des Clients werden in diesem Beispiel an einem globalen Service Pool gerichtet, der als Proxy agiert und den Serviceaufruf an eine Service Instanz weiterleitet, die über freie Ressourcen zur Beantwortung der Anfrage verfügt.

Die Ergebnisse der Skalierbarkeitsanalyse sind in Abbildung 5.10 dargestellt. Um mögliche Zacken in der Beantwortungszeit zu dämpfen, wurden jeweils $10$ Experimente für jede Anzahl von $n$ parallelen Anfragen durchgeführt und die Mittelwerte berechnet. Für $n = 10, 100, 1000$ parallele Anfragen zeigen alle drei Implementierungsalternativen ein ähnliches, linear skalierendes Verhalten. Dieses Verhalten ändert sich allerdings für $n = 10000$ parallele Serviceanfragen. Hier skaliert nur die Implementierung, bei der die Services auf Basis ihrer nicht-funktionalen Eigenschaften ausgewählt werden (2), linear. Bei den anderen beiden Alternativen verschlechtert sich die Skalierbarkeit. Daher wurden weitere Experimente für diese beiden Implementierungsalternativen für $n = 2000, 3000, 4000, 5000$ parallele Anfragen durchgeführt. Im Fall von $2000 \leq n \leq 4000$ zeigen die beiden Alternativen dabei auch noch ein annähernd linear skalierendes Verhalten. Dieses ändert

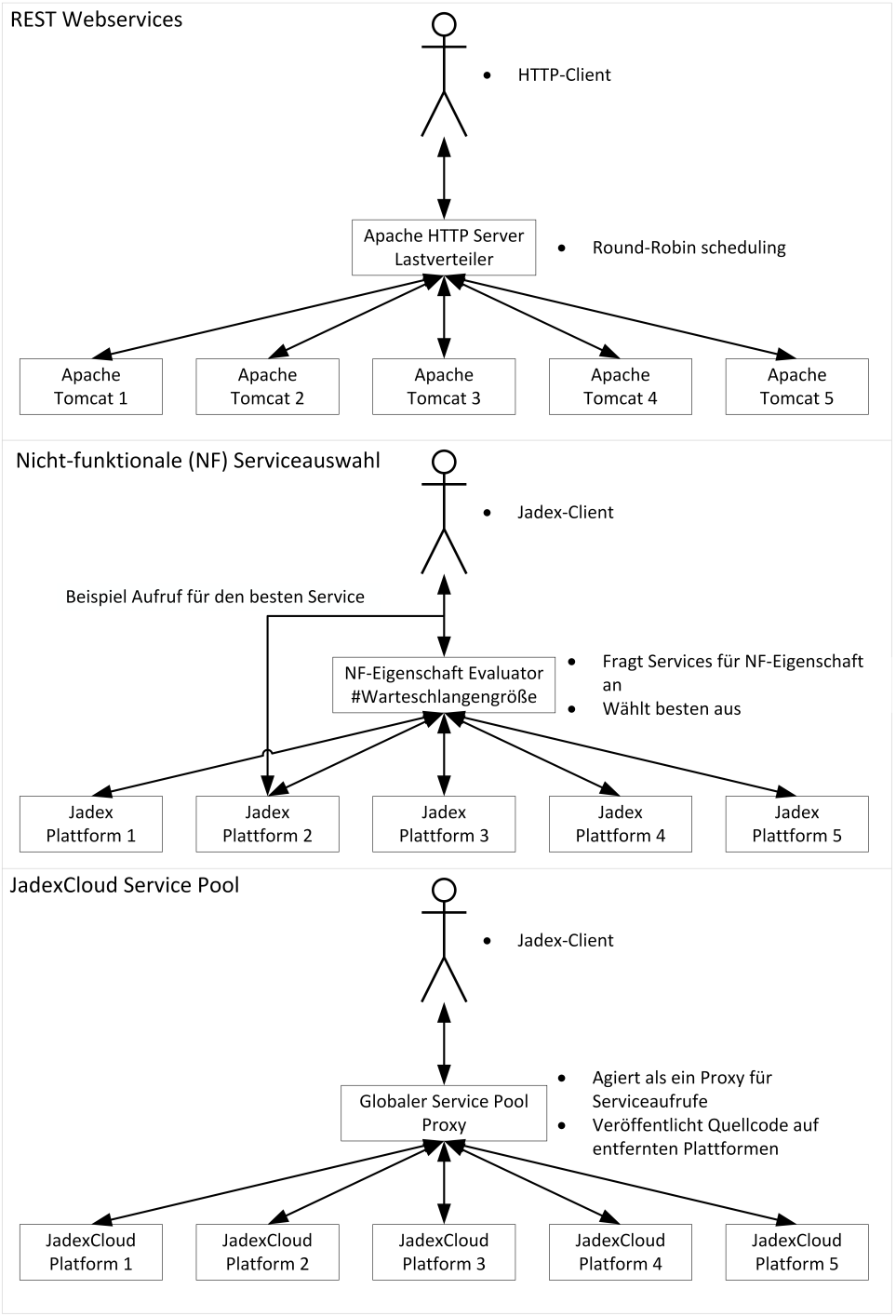



**Abbildung 5.9.:** Konfigurationsbeispiele für drei exemplarische Implementierungsalternativen für den Simulation als ein Service Ansatz - nach (PDR15c).

sich, sobald $n \geq 5000$ wird. Eine mögliche Erklärung für diese Abweichung bzgl. der Skalierbarkeit der drei Ansätze ist es, dass die Lastverteiler in Form des *Apache HTTP Servers* (1) und des *Jadex Service Pools* (3) gesättigt werden. Wenn die Lastverteiler eine große Anzahl von parallelen Anfragen bearbeiten müssen, können diese selber an ihre Verarbeitungsgrenzen kommen und dabei zum Flaschenhals im System werden. Im Fall der Serviceauswahl auf Basis der nicht-funktionalen Eigenschaften wird die Auslastungsrate der Dienste von dem anfragenden Client selber übernommen. Daher liegt der entsprechende Berechnungsaufwand hier dezentral auf Seiten des Clients und wird somit auf diese verteilt.

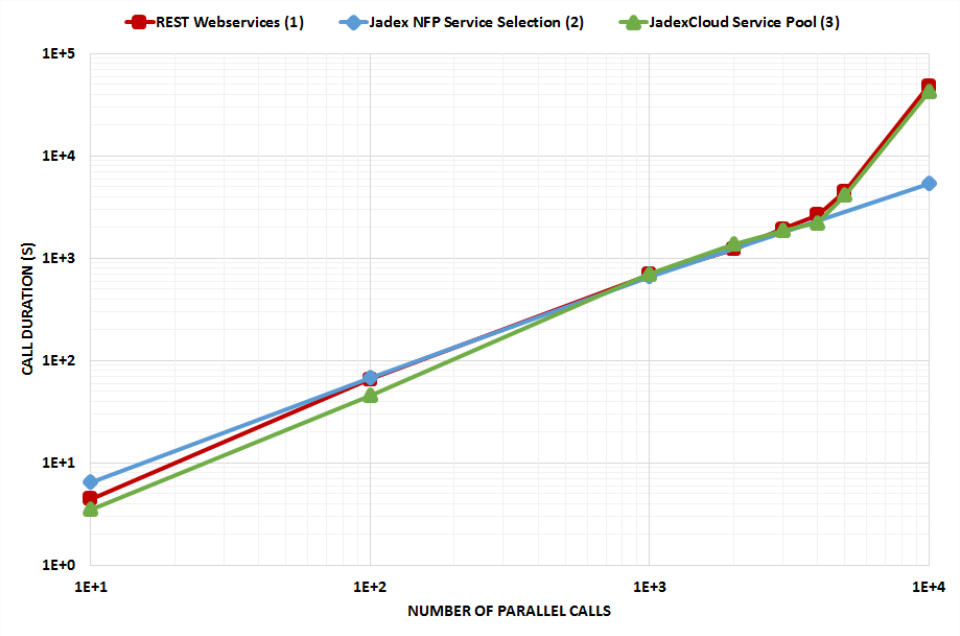



**Abbildung 5.10.:** Ergebnisse der Skalierbarkeitsanalyse - nach (PDR15c).

Abschließend lässt sich feststellen, dass es grundsätzlich verschiedene Möglichkeiten gibt, den Ansatz, Simulationen in Form eines Services bereitzustellen, zu realisieren. Abhängig davon, ob es um die Einbindung einer bestehenden, ggf. monolithischen Simulation oder die Neuentwicklung einer bereits als verteiltes System konzipierten Simulation handelt, bieten sich dabei unterschiedliche Lösungen und Konzepte aus dem Umfeld des *Cloud Computing* an. Im Kontext der Service- und Komponenten-basierten Systeme gibt es weitere Lösungen, welche die diese Konzepte sinnvoll ergänzen und basierend auf den bei der Entwicklung zu identifizierenden Anforderungen ausgewählt werden sollten.

### 5.2.5. Daten-adaptive Simulation als Service

In Abschnitt 5.1.2 wurde eine komponentenbasierte Architektur für ein servicebasiertes Simulationsframework zur Unterstützung von adaptiven Co-Simulationen vorgestellt. Dieses Konzept kann u.a. dazu verwendet werden,

adaptive Simulationen als Dienste in einer Cloud-Infrastruktur anzubieten. Eine entsprechende Lösung wurde in [PDR15a] im Bezug auf eine Daten-adaptive Simulation erarbeitet. Das dort entwickelte System ist adaptiv im Bezug auf seine Daten, das bedeutet, dass der Daten- oder Wissensraum der Anwendung bei Bedarf gefüllt wird. Verfügt das System bereits über die benötigten Daten, um eine spezifische Simulationsanfrage zu beantworten, werden diese sofort zurückgegeben. Sollten für die Parameter der Anfrage keine entsprechenden Ergebnisse vorliegen, wird die notwendige Simulation ausgeführt und der Daten- bzw. Wissensraum mit den Ergebnissen angereichert. Bei diesem Ansatz verwaltet ein Simulationssystem die Anfragen für eine spezifische Simulationskomponente. Eine solche Anfrage enthält ein variables Tupel von Kriterien, die die Ausführung der Simulationskomponente parametrisieren. Das Simulationssystem leitet diese Anfrage an den Daten- bzw. Wissensraum weiter, wo die Ergebnisse der Simulation für unterschiedliche Parameter-Tupel gespeichert sind. Dieser Wissensraum ist aktiv, in dem Sinne, dass er in Lage ist, eigenständig zu entscheiden, ob die Ausführung der Simulationskomponente gestartet werden soll, wenn keine Daten für das gegebene Parameter-Tupel vorliegen. In diesem Fall wird die Simulationskomponente ausgeführt und die Ergebnisse werden im Wissensraum gespeichert, bevor sie an den aufrufenden Client zurückgegeben werden. Weitere Simulationsanfragen für das gleiche Parameter-Tupel erfordern somit keine erneute Ausführung der Simulationskomponente, sondern können direkt mit dem bestehenden Wissen beantwortet werden. Abbildung 5.11 illustriert den Daten-adaptiven Ansatz.

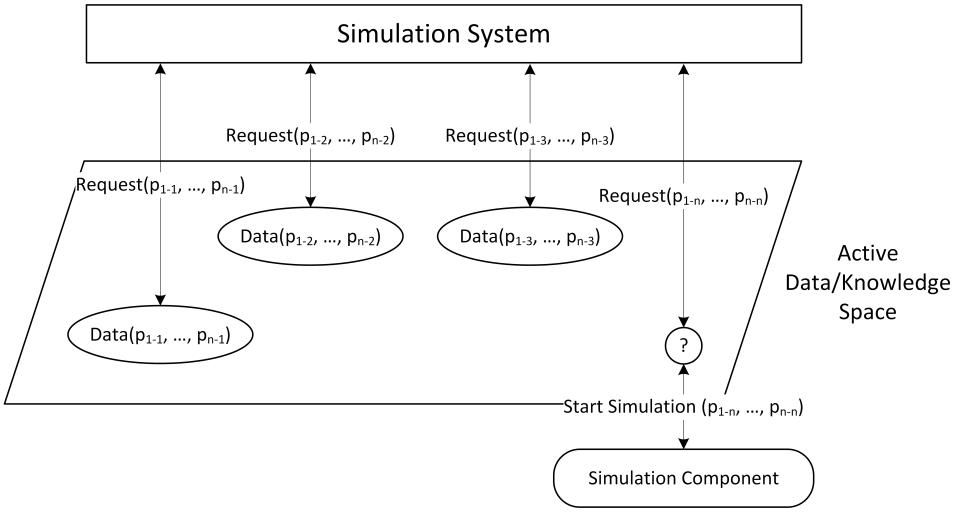



**Abbildung 5.11.:** Daten-adaptiver *Simulation as a Service* Ansatz - aus (PDR15a, S. 11).

Abbildung 5.12 zeigt in diesem Zusammenhang, wie die Architektur eines solchen Daten-adaptiven Simulationsservicesystems aussehen kann. Die Funktionalität der Simulation wird dabei als Serviceschnittstelle bereitgestellt. Dieser Service kapselt die eigentliche Implementierung der Simulationskomponente und des Daten-adaptiven Verhaltens. Basierend auf dem Daten-

adaptiven Ansatz prüft der Simulationsservice bei Aufruf zunächst, ob der Daten- bzw. Wissensraum (hier realisiert in Form einer Datenbank) Ergebnisse für die entsprechenden Anfrageparameter enthält (1). Wenn dies zutrifft, werden diese direkt an den anfragenden Client zurückgegeben. Wenn keine Ergebnisdaten vorliegen, führt der Service die Simulationskomponente aus (2) und speichert die Ergebnisse in der Datenbank (3). In diesem Fall wird das aktive Verhalten des Daten- bzw. Wissensraums als Teil des Simulationsservice realisiert und die Datenbank bietet nur Funktionen zur Aufbewahrung von Daten. Eine andere mögliche Realisierung könnte in Form einer aktiven Datenbank erfolgen, die dieses Verhalten z.B. in Form von *Stored Procedures* bereitstellt.

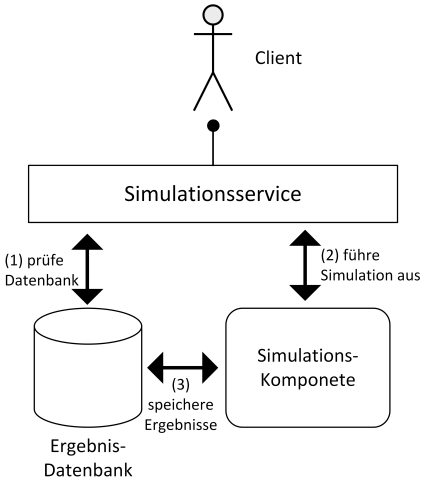



**Abbildung 5.12.:** Daten-adaptive Simulationsservice Architektur - nach (PDR15a).

## 5.3. Zusammenfassung

In diesem Kapitel wurde ein Überblick über skalierbare, verteilte und adaptive Simulationssysteme gegeben. Simulationen spielen u.a. eine entscheidende Rolle bei der Validierung und Evaluation von selbstorganisierenden Verhalten in verteilten Systemen (vgl. u.a. [Vil13]), profitieren dabei aber auch selber von adaptiven oder selbstorganisierenden Eigenschaften und der Fähigkeit zur Koordination. Vom besonderen Interesse sind im Rahmen dieser Dissertation dabei Co-Simulationssysteme. Hierbei formen unterschiedliche Simulationssysteme gemeinsam ein gekoppeltes Problem, wobei die Ausführung der Simulation i.d.R. verteilt stattfindet. Ein wichtiges, aktuelles Anwendungsgebiet für Co-Simulationen ist die Planung und Realisierung sogenannter *Smart Cities*. Hierbei handelt es sich um einen Sammelbegriff für gesamtheitliche Entwicklungskonzepte, die darauf abzielen, Städte in Zukunft effizienter, technologisch fortschrittlicher und umweltverträglicher zu gestalten.

Ein entscheidender Aspekt bei der Realisierung von Co-Simulationen, z.B. im Rahmen der Simulation von solchen intelligenten Städten, ist die Interoperabilität der unterschiedlichen Simulationssysteme und -modelle. Daher

wurden in diesem Kapitel zunächst existierende Ansätze zur Herstellung der Interoperabilität vorgestellt. Anschließend wurde eine Servicekomponenten-basierte Architektur vorgestellt, die darauf abzielt, sowohl mikroskopische als auch makroskopische Simulationen im Rahmen einer Co-Simulation miteinander zu verbinden. Das Ziel der entwickelten Architektur ist es dabei, einen integrativen Ansatz zu schaffen, um unterschiedliche Simulationskonzepte, die oftmals auf unterschiedlichen Zeitskalen und -modellen sowie Werkzeugen basieren, zu verbinden. Als Integrationsansatz wurde dazu die Bereitstellung der unterschiedlichen (Teil-)Simulationen als Dienste im Rahmen einer serviceorientierten Architektur gewählt. Ein wichtiger Aspekt, der den präsentierten Ansatz von anderen existierenden Ansätzen abgrenzt, ist die Verwendung eines dezentralen, adaptiven Ansatzes. Dabei befindet sich die Logik, die bestimmt, welche (Teil-)Simulationen für ein bestimmtes Simulationsszenario ausgeführt werden sollen, nicht in einer zentralen Komponente, sondern ist Teil einer dezentralen, adaptiven Komponente, die jeweils für die Ausführung und Überwachung einer Simulationskomponente verantwortlich und dieser zugeordnet ist. Dieses Konzept wurde dabei im Rahmen dieses Kapitel anhand eines Beispiel zur skalierbaren Integration von 4GL-Simulationsmodellen verdeutlicht. Im Rahmen dieses Beispiel wurde ebenfalls ein Ausblick darüber gegeben, wie der erdachte Integrationsansatz verwendet werden kann, um kombinierte Hard- und Software Simulationen zu realisieren. Gerade im Bereich der Entwicklung von Anwendungen und Kontrollstrukturen für intelligente Energienetze (*Smart Grids*), die im Zuge der aktuellen Energiewende hin zu erneuerbaren, verteilten Energieressourcen immer mehr an Bedeutung gewinnen, sind solche cyber-physikalischen Systeme zunehmend von großer Bedeutung.

Basierend auf diesem Servicekomponenten-basierten Ansatz wurde das Konzept, Simulationen als einen Dienst im Rahmen einer *Cloud Computing* Infrastruktur anzubieten, beschrieben. Hierbei handelt es sich um einen spezifischen Ansatz, der sowohl die Interoperabilität von unterschiedlichen Simulationssystemen und -komponenten durch deren Bereitstellung als Dienste sicherstellt als auch auf deren skalierbare Ausführung im Rahmen einer *Cloud Computing* Umgebung abzielt. Dazu wurde zunächst ein Überblick über aktuelle *Cloud Computing* Technologien gegeben, bevor auf existierende Ansätze zur Bereitstellung von Simulationen als Dienste eingegangen wurde. Schließlich wurde auf ein eigenes Konzept zur Bereitstellung solcher Simulationsdienste innerhalb einer *Cloud Computing* Infrastruktur eingegangen und beispielhaft drei unterschiedliche Implementierungsalternativen vorgestellt, die im Rahmen dieser Dissertation entwickelt wurden. Abschließend wurde mit dem Ansatz, eine Daten-adaptive Simulation als einen Service anzubieten, ein Beispiel gegeben, dass aufzeigt, wie eine Simulation sowohl um ein adaptives Verhalten, in diesem Fall im Bezug auf den Daten- bzw. Wissensraum der Anwendung, als auch um eine einheitliche Aufrufsemantik in Form eines Dienstes erweitert werden kann.

Abschließend lässt sich feststellen, dass Simulationen nicht nur Anwendung finden können, um selbstorganisierende Systeme im Bezug auf ihr selbstorganisierendes Verhalten zu evaluieren, sondern auch ihrerseits von adaptiven Verhalten profitieren können. Dabei wurde deutlich, dass die Interoperabilität von unterschiedlichen Simulationsmodellen und -konzepten von entscheidender Bedeutung ist. Außerdem wurde deutlich, dass gerade im Bezug auf Simulationen, die darauf abzielen, zukünftige Entwicklungen im Bereich der intelligenten Städte oder Energienetze zu unterstützen, die Integration unterschiedlicher Simulationswerkzeuge entscheidend ist. Im Bezug auf die Konstruktion von selbstorganisierenden Verhalten bedeutet dies, dass es in diesem Umfeld nicht ausreicht, Ansätze wie den *Coordination Space* (siehe Kapitel 4), der in seiner Anwendbarkeit auf die Koordination von Multiagentensystemen beschränkt ist, zu verwenden. Vielmehr erfordert es sowohl im Hinblick auf die Realisierung von cyber-physikalischen Systemen als auch im Rahmen der Simulation und des Testens von dezentralen Kontrollstrukturen für *Smart Grids* einen Koordinationsansatz, der verteilte Systeme im Allgemeinen und nicht nur Multiagentensysteme im Speziellen unterstützt.

# 6. Dezentrale Koordination in verteilten Systemen

In diesem Kapitel wird ein neuartiger Ansatz zur Realisierung dezentraler Koordination in verteilten Systemen vorgestellt. Die Fähigkeit zur dezentralen Koordination ist ein integraler Bestandteil selbstorganisierender Systeme (vgl. Abschnitt 2.3). Um in einem selbstorganisierenden System ein kohärentes, adaptives Verhalten erzielen zu können, müssen die beteiligten Komponenten und ihre Aktivitäten in sinnvoller Weise koordiniert werden. In Abschnitt 4.3 wurde daher mit dem Konstruktionsansatz des *Coordination Space* eine Lösung präsentiert, welche die dezentrale Koordination von Multiagentensystemen unterstützt und somit die Konstruktion von selbstorganisierenden Systemen auf Basis von Multiagenten-Technologie ermöglicht. Im vorangegangen Kapitel wurde ein Überblick über skalierbare, verteilte und adaptive Co-Simulationssysteme gegeben. Dabei wurde u.a. deutlich, dass eine essentielle Herausforderung bei der Realisierung solcher Systeme die Heterogenität der verwendeten Technologien und Plattformen ist, die für neue Anforderungen im Bereich der Interoperabilität sorgt. Gleichwohl gibt es aber auch gute Gründe, unterschiedliche Technologien und Ansätze zur Modellbildung und Simulation einzusetzen (vgl. u.a. [SWA+13]). Ein Ansatz wie der zuvor beschriebene *Coordination Space*, der eine Lösung speziell für Multiagentensysteme liefert, ist daher nicht ausreichend, wenn es um die Koordination von heterogenen Systemen, z.B. im Umfeld von Co-Simulationen, geht.

In diesem Kapitel wird daher ein neuartiger Ansatz beschrieben, der einige Konzepte aus der Integrationsarchitektur des *Coordination Space* und dessen Vorarbeiten aufgreift und diese für ein allgemeineres Konzept, das die Konstruktion selbstorganisierender Systeme unabhängig von der verwendeten Systemarchitektur erlaubt, adaptiert. Durch den von der Systemarchitektur unabhängigen Ansatz wird u.a. die Koordination von Komponenten in einem Co-Simulationssystem ermöglicht. Der Aufbau dieses Kapitel gliedert sich dabei wie folgt: Zunächst wird ein Konzept beschrieben, das die dezentrale Koordination in verteilten Systemen, die auf Basis des *Aktiven Komponenten* Konstruktionsansatz realisiert sind, unterstützt. Dieses Konzept kann als Zwischenschritt verstanden werden, der den Übergang von der Koordination von Multiagentensystemen im Speziellen hin zu verteilten Systemen im Allgemeinen vorbereitet. Im darauffolgenden Abschnitt wird anschließend das im Rahmen dieser Dissertation entwickelte Rahmenwerk zur dezentralen Koordination in verteilten Systemen (*DeCoF*, *„Decentralized Coordination Framework“*) im Detail beschrieben.

## 6.1. Dezentrale Koordination von Aktiven Komponenten

Im Rahmen dieser Dissertation wurde eine Middleware entworfen, die es ermöglicht, sogenannte *Aktive Komponenten* zu koordinieren und somit Anwendungen, die auf diesem Paradigma basieren, mit selbstorganisierenden Verhalten auszustatten [PDR15b]. Der Ansatz basiert dabei konzeptionell auf dem Konstruktionsansatz des *Coordination Space* (vgl. Abschnitt 4.3) und dessen Vorarbeiten (*DeCoMAS* [SR09a]). *Aktive Komponenten* kombinieren das autonome Verhalten von Softwareagenten mit dem Dienstanbieter-Paradigma aus dem Umfeld *der Service Component Architecture* (SCA). Der konzeptuelle Ansatz basiert dabei auf zwei Annahmen bezüglich der Konstruktion von verteilten Systemen: Zunächst wird aus dem Bereich der agenten-orientierten Softwareentwicklung angenommen, dass die Modellierung eines Systems basierend auf aktiven und passiven Entitäten besser geeignet ist, um die Realität abzubilden, als wenn dieses durch die Abbildung auf Objekt- oder Komponentenorientierte Systeme geschieht. Die zweite Annahme bezieht sich auf die Serviceorientierung. Es ist von Vorteil, Systeme auf Basis von Entitäten zu realisieren, welche Dienste mit klar abgetrennter Funktionalität bereitstellen, auswählen und benutzen. Basierend auf diesen beiden Annahmen adaptiert der *Aktive Komponenten* Ansatz eine agenten-orientierte Sichtweise im Hinblick auf das autonome und nebenläufige Verhalten der handelnden Entitäten und kombiniert diese mit einer serviceorientierten Perspektive (vgl. [BP12c]). Das zugrundeliegende Konzept wurde im Rahmen der *Jadex Active Components* Plattform [BP12c] umgesetzt. Die Plattform ermöglicht es somit, auf Basis dieses Ansatzes verteilte Systeme zu realisieren und bildet im Folgenden die Grundlage für die dezentrale Koordination von Aktiven Komponenten. Für eine ausführlichere Beschreibung des Konzepts der *Aktiven Komponenten* wird an dieser Stelle nochmals auf Abschnitt 3.1.3 verwiesen.

### 6.1.1. Entwicklungsansatz

Der Entwicklungsansatz zur Konstruktion von dezentraler Koordination für Aktive Komponenten basiert stark auf dem autonomen Verhalten der Komponenten und ihrem serviceorientierten Kommunikationsparadigma. Im Folgenden wird daher zunächst die Zielsetzung der in [PDR15b] vorgestellten Middleware beschrieben, bevor das autonome Verhalten der Aktiven Komponenten und die serviceorientierte Kommunikation näher betrachtet werden. Schließlich wird darauf eingegangen, wie diese Konzepte verwendet werden, um dezentrale Koordinationsstrukturen zu realisieren.

#### Zielsetzung

Die grundlegende Zielsetzung der Middleware ist es, verteilte Anwendungen, die auf dem Konzept der Aktiven Komponenten basieren, mit selbstorganisierenden Verhalten auszustatten. Der Ansatz folgt dabei der strikten Trennung zwischen der Geschäfts- und Koordinationslogik nach [GC92], wie sie bereits in

vorherigen Teilen dieser Arbeit gefordert wurde. Zur Realisierung des selbstorganisierenden Verhaltens wird die Fähigkeit zur Koordination benötigt, um z.B. Feedback-Schleifen zu realisieren (siehe Abschnitt 3.1.1). Der Ansatz basiert dabei auf dem Konzept, dezentrale Koordinationsmechanismen zu verwenden, um die benötigte dezentrale Kontrolle zu realisieren (vgl. Abschnitt 2.3.3). Ähnlich wie bei dem zugrundeliegenden *Coordination Space* [VSL+10] und *DeCoMAS* [SR09a] Ansätzen wird dazu eine domänenspezifische Sprache verwendet, um die Koordinationsmechanismen deklarativ zu beschreiben und somit eine lose Kopplung zur Implementationsplattform zu erreichen.

### Autonomes Verhalten von Aktiven Komponenten

Das Verhalten von Aktiven Komponenten wird durch sogenannte *Component Steps* modelliert. Diese kapseln das Verhalten der Aktiven Komponenten in spezifische Ausführungsschritte. Aus einer technischen Sichtweise heraus ist ein solcher *Component Step* eine Klasse, die das entsprechende Interface implementiert. Die Ausführung eines *Component Steps* kann entweder durch eine externe Quelle (z.B. eine andere Aktive Komponente oder ein Service, den die Komponente anbietet) gestartet werden oder intern innerhalb eines anderen *Component Steps* erfolgen. Die Ausführung eines *Component Steps* kann also von unterschiedlichen Quellen auf unterschiedliche Art und Weise ausgelöst werden. Außerdem kann nebenläufiges Verhalten in einer Aktiven Komponenten durch die parallele Ausführung von *Component Steps* realisiert werden. Dabei werden die notwendigen Scheduling-Mechanismen von der *Jadex* Ausführungsplattform angeboten und vorgenommen. Das Konzept, wie das Verhalten einer Aktiven Komponenten aus mehreren *Component Steps* zusammengesetzt gesetzt werden kann, wird in Abbildung 6.1 verdeutlicht.

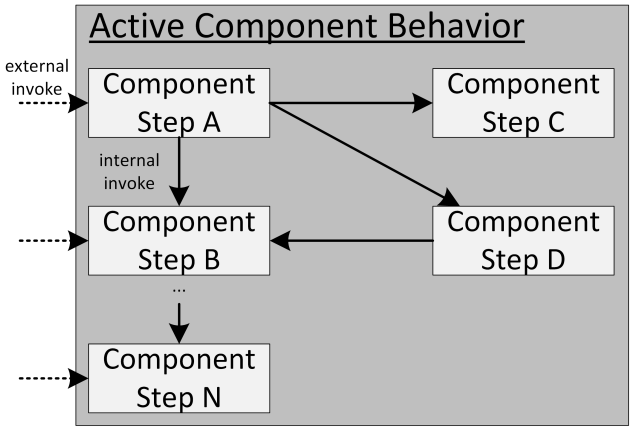



**Abbildung 6.1.:** Verhalten einer Aktiven Komponente, das sich aus mehreren *Component Steps* zusammensetzt - aus (PDR15b, S. 3).

### Serviceorientierte Kommunikation

Basierend auf dem SCA-Konzept bieten Aktive Komponenten bestimmte Services an und benötigen wiederum andere. Durch die Service-Metapher ist es dabei möglich, unterschiedliche Interaktionsmuster abzubilden. Dabei können sowohl Interaktionen basierend auf dem klassischen Methodenaufruf-Paradigma realisiert werden als auch basierend auf einer serviceorientierten Interaktion. Im Fall des Methodenaufruf-Paradigmas, wie es z.B. in der Objektorientierung bekannt ist, kann der Service als Methode betrachtet werden, die von einer bestimmten Komponenten angeboten wird. Im Fall einer serviceorientierten Interaktion spielt die Identität des Serviceanbieters keine Rolle, hier ist nur die angebotene Funktionalität des Dienstes von Bedeutung. Zusätzlich ermöglicht es der sogenannte *Awareness*-Mechanismus der *Jadex* Plattform, entfernte Plattformen, die auf anderen Knoten im Netzwerk ausgeführt werden, automatisch zu detektieren. Somit wird automatisiert ein Overlay-Netzwerk gebildet, das es ermöglicht, dass Komponenten auf die Dienste, die andere Komponenten auf entfernten Plattformen anbieten, zugreifen können. Diese Overlay-Funktionalität wird den Komponenten transparent angeboten und erlaubt es, unterschiedliche Bereiche anzugeben, in denen nach bestimmten Diensten gesucht werden soll (z.B. innerhalb der lokalen Plattform oder im Kontext einer verteilten Anwendung).

### Konstruktion dezentraler Kontrollstrukturen

Der erste Schritt bei der Konstruktion von dezentralen Kontrollstrukturen ist es, zu identifizieren, wann die teilnehmenden Komponenten koordiniert werden müssen. Da die Koordinationsmiddleware auf dem Konzept der Aktiven Komponenten beruht, muss entsprechend die Ausführung dieser Komponenten observiert werden, um Zustände zu detektieren, in denen Koordination erforderlich ist. Im Gegensatz zu BDI-Agenten, bei denen z.B. das Auftreten bestimmter *Ziele*, die Ausführung bestimmter *Pläne* oder der Wert bestimmter *Beliefs* von außen observiert werden kann, um daraus den Zustand des Agenten zu bestimmen, kann bei Aktiven Komponenten nur die Ausführung der *Component Steps* observiert werden, um daraus Informationen über den Zustand der Komponente abzuleiten. Da das globale Verhalten einer Aktiven Komponenten aus mehreren *Component Steps* zusammengesetzt ist, kann die Ausführung eines spezifischen *Component Steps* auf einen bestimmten Zustand der Aktiven Komponente abgebildet werden. Daher gilt es, bei der Konstruktion dezentraler Kontroll- bzw. Koordinationsstrukturen für Aktive Komponenten zunächst die relevanten Zustände der Komponente zu identifizieren und diese auf die Ausführung entsprechender *Component Steps* abzubilden.

Im zweiten Schritt gilt es dann zu identifizieren, welche adaptive Aktion ausgeführt werden soll, wenn eine Komponente einen bestimmten Zustand erreicht. Diese adaptiven Aktionen werden, wie alle anderen Verhalten in einer Aktiven Komponente auch, auf die Ausführung eines bestimmten *Component Steps* abgebildet. Um also das Verhalten bzw. die Aktionen von Aktiv-

en Komponenten zu kontrollieren bzw. zu koordinieren, ist es erforderlich, die Ausführung ihrer *Component Steps* zu koordinieren. Dazu wird eine Beschreibungssprache benötigt, die es ermöglicht, zu beschreiben, welche Aktionen in welchem Typ vom Komponenten auf die Ausführung von welchen anderen Aktionen in welchen anderen Typen von Komponenten abgebildet werden sollen. Die für die Koordinationmiddleware realisierte Beschreibungssprache wird in Unterabschnitt 6.1.3 im Detail beschrieben. Grundsätzlich wird ein Rollenkonzept verwendet, das spezifische Typen von Aktiven Komponenten auf bestimmte Rollen abbildet. Basierend auf der Beschreibung dieser Rollen kann spezifiziert werden, welcher *Component Step*, der in einer bestimmten Rolle ausgeführt wird, die Ausführung welches anderen *Component Steps* in den Aktiven Komponenten einer anderen (oder derselben) Rolle auslösen soll. Zusätzlich erlaubt es die Beschreibungssprache anzugeben, welche Art von Kontrolldaten dabei von einem *Component Step* auf den anderen übertragen werden sollen.

Der zweite Aspekt bzgl. der Koordination von Aktiven Komponenten und der Ausführung von *Component Steps* betrifft die serviceorientierte Kommunikation. Da die in [PDR15b] vorgestellte Middleware auf einem serviceorientierten Kommunikationsansatz basiert, muss jede Aktive Komponente, die koordinierbar sein soll, einen entsprechenden *Koordinationsservice* anbieten. Dieser Service bietet Methoden an, um die benötigten Kontrolldaten zu empfangen und die Ausführung von spezifizierten *Component Steps* zu starten. Damit die Koordinationsmiddleware in der Lage ist, die Koordinationsservices der entsprechenden Aktiven Komponenten auswählen, werden sogenannte *Coordination Selectors* (Koordinationsselektoren) verwendet. Aus implementationstechnischer Sicht verwenden diese Selektoren die *Awareness* Funktionalität der *Jadex*-Plattform, um entfernte Koordinationsservices aufzufinden und die passenden basierend auf den gegebenen Koordinationsinformationen zu selektieren. Die Aufgabe der Selektoren entspricht damit in etwa derer der Koordinationsmedien aus dem *Coordination Space* und *DeCoMAS* Ansätzen. Sie sind somit für die Verbreitung der Koordinationsinformationen verantwortlich.

### 6.1.2. Architektur

Die Architektur der Koordinationsmiddleware basiert auf einer klaren Trennung zwischen der Anwendungsschicht und der darunterliegenden Koordinationsschicht. Die Anwendungsschicht bildet eine verteilte Anwendung ab, die z.B. aus mehreren Softwareagenten bestehen kann. Softwareagenten sind eine mögliche Ausprägung von Aktiven Komponenten. Eines der Ziele der vorgeschlagenen Koordinationsmiddleware ist es, auch existierende Anwendungen mit dezentralen Koordinationsmechanismen ausstatten zu können. Dieses wird erreicht, indem die Koordination in eine eigenständige Schicht ausgelagert wird. In Bezug auf die lose Kopplung zwischen der Anwendungs- und der Koordinationslogik wird dazu jede Aktive Komponen-

ten mit einem speziellen Koordinationsservice ausgestattet. Dieser Service stellt eine Schnittstelle bereit, um das Verhalten der Aktiven Komponente zu kontrollieren. Immer, wenn der Koordinationsservice einer Komponente aufgerufen wird, erhält die Komponente entsprechende Koordinationsinformationen. Das sind Informationen, die Anweisungen bzgl. der *Component Steps*, die auf der Aktiven Komponente ausgeführt werden sollen, enthalten, um somit das Verhalten der Komponente zu kontrollieren. Abbildung 6.2 stellt die entsprechende Schichtenarchitektur der dezentralen Koordinationsmiddleware dar.

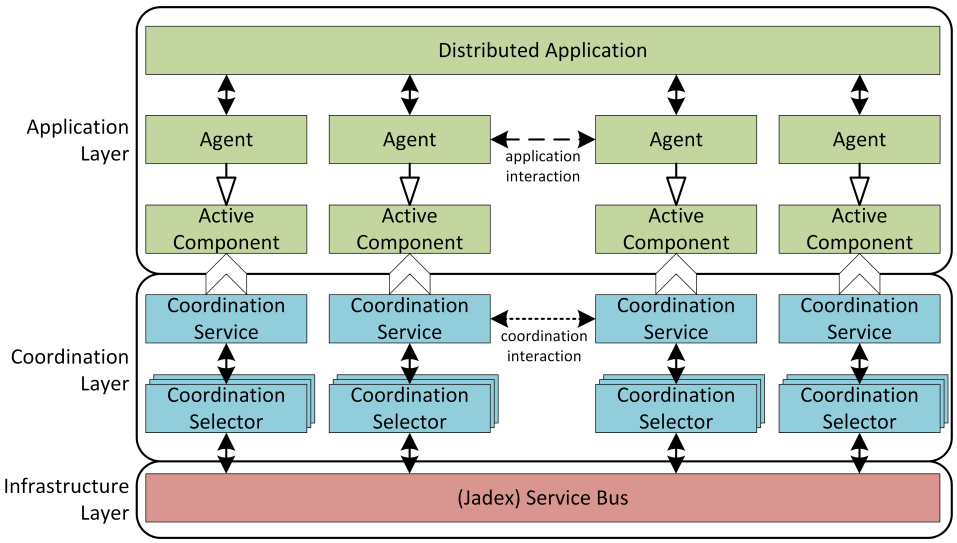



**Abbildung 6.2.:** Schichtenarchitektur der dezentralen Koordinationsmiddleware für Aktive Komponenten - nach (PDR15b, S. 4).

Ein Koordinationsservice, wie er zuvor beschrieben wurde, besteht aus zwei Komponenten; dem sogenannten *Component Executor* und dem *Component Monitor*. Der *Component Executor* ist dafür verantwortlich, die Ausführung der *Component Steps* auf der Aktiven Komponente einzuplanen. Dazu wird der *Executor* von Koordinationsservice aufgerufen, wenn über diese Schnittstelle Koordinationsinformationen empfangen werden. Die Aufgabe des *Component Monitor* ist es, die Ausführung von *Component Steps* auf der Aktiven Komponente zu observieren. Wenn dabei die Ausführung eines *Component Steps*, der für die Koordination relevant ist, beobachtet wird, löst der Monitor die Koordinationsinformationsausbreitung aus. Wie genau diese Daten verbreitet werden, unterliegt dann der Kontrolle des entsprechenden *Coordination Selectors*. Dieser selektiert die passenden, entfernten Koordinationsservices und ruft diese anschließend auf, sodass die entsprechenden Aktiven Komponenten koordiniert werden können. Abbildung 6.3 liefert eine Detailansicht einer Aktiven Komponenten und ihrer Koordinationserweiterung.

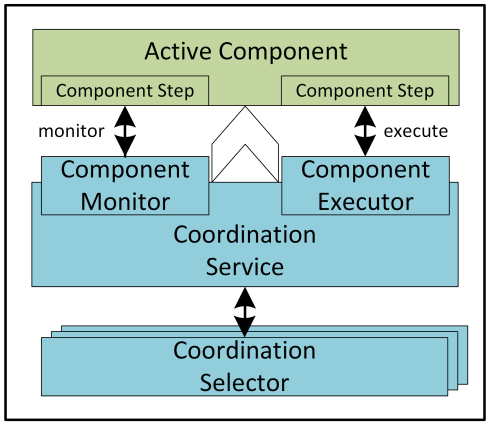



**Abbildung 6.3.:** Detailansicht einer Aktiven Komponente und ihrer Koordinationserweiterung - nach (PDR15b, S. 4).

### 6.1.3. Implementierung

Die Middleware zur Konstruktion von dezentraler Kontrolle in selbstorganisierenden System wurde als Erweiterung der *Jadex Active Components* Plattform realisiert. Dieser Unterabschnitt liefert einen Überblick über die Beschreibungssprache, die entwickelt wurde, um den Koordinationsprozess deklarativ beschreiben zu können und stellt die einzelnen Implementierungsschritte vor, die ausgeführt werden müssen, um eine Anwendung mit der Fähigkeit zur Koordination zu erweitern.

#### Koordinationsbeschreibungssprache

Um zu beschreiben, wie die dezentrale Koordination einer verteilten Anwendung erfolgen soll, wurde eine XML-basierte Konfigurationssprache entwickelt, die als Nachfolger der *MASDynamics* Sprache [SR09b] angesehen werden kann. Dabei wurde die neu entwickelte Koordinationsbeschreibungssprache an einigen Stellen vereinfacht und ein Kontextwechsel von der Koordination von Softwareagenten hin zu Aktiven Komponenten vollzogen. Die Koordinationsbeschreibung (`coordination-description`) besteht dabei grundsätzlich aus zwei Teilen: Der erste Teil beschreibt die Rollen (`role-definitions`). Hier werden Aktive Komponenten gemäß ihrem Typ bestimmten Rollen zugeordnet. Dadurch ist es möglich, das gemeinsame Verhalten von mehreren unterschiedlichen Komponenten(-typen) auf abstrakterem Level zu beschreiben. Der zweite Teil beschreibt die Definition der dezentralen Koordinationsverbindungen (`deco-link-definition`) zwischen den unterschiedlichen Rollen. Eine solche Koordinationsverbindung (`deco-link`) besteht wiederum aus drei Teilen: Der erste Teil gibt Auskunft über den *Component Step*, dessen Ausführung den Koordinationsprozess auslöst (`from`). Entsprechend beschreibt der zweite Teil, welcher *Component Step* von der Koordinationsmiddleware in der entsprechenden Aktiven Komponente ausgeführt werden soll, wenn diese die Koordinations-

informationen über ihre Koordinationsschnittstelle erhält (`to`). Der letzte Teil der Beschreibung (`selector`) betrifft die Konfiguration des *Coordination Selector*. Hier wird eine Java-Klasse angegeben, die das entsprechende `ICoordinationSelector` Interface implementiert. Zusätzlich kann der Selektor mit optionalen Parametern weiter konfiguriert werden. Die XML Schema Definition der Beschreibungssprache ist in Quelltext A.2 aufgeführt.

### Implementierungsschritte

Um nun eine Anwendung mit der Fähigkeit zur dezentralen Koordination auszustatten, sind bei der Verwendung der Koordinationsmiddleware folgende Implementierungsschritte notwendig. Die Reihenfolge ist dabei nicht zwingend festgelegt, wird allerdings empfohlen:

1. Identifizierung der für die Koordination relevanten Typen von Aktiven Komponenten und ihre Abbildung auf die entsprechenden Rollen.

2. Identifizierung der *Component Steps*, die für die Koordination verwendet werden sollen.

3. Diese Informationen sollten nun dazu verwendet werden, um mit der Koordinationssprache zu beschreiben, wie die Middleware die Applikation koordinieren soll.

4. Implementierung der Koordinationsselektoren, die innerhalb der Koordinationsbeschreibung verwendet wurden. Dazu müssen die entsprechenden Java-Klasse das `ICoordinationSelector` Interface implementieren. Diese Schnittstelle gibt die Signatur der Selektionsmethode vor. Die genaue Methode, nach der die entsprechenden Koordinationsservices selektiert werden, kann u.a. von der Anwendungslogik oder der Netzwerktopologie abhängen. Ergänzend dazu bietet die Middleware bereits fertige Lösungen für Selektoren, die auf einer *Unicast* oder *Broadcast* Lösung basieren.

5. Der letzte Implementierungsschritt befasst sich mit plattformspezifischen Eigenschaften der *Jadex Active Components* Plattform. Hier wird die Verwendung von Annotationen unterstützt, die beschreiben, welche Dienste eine Aktive Komponente benötigt und bereitstellt. Die Middleware verwendet die `@ProvidedService` Annotation, um Komponenten mit einem Koordinationsservice auszustatten. Die Annotation wird dabei mit dem Pfad zur XML-Datei, welche die deklarative Koordinationsbeschreibung beinhaltet, erweitert. Wird nun die Anwendung gestartet, fügt die Plattform allen Aktiven Komponenten mit der entsprechenden Annotation automatisch den Koordinationsservice hinzu. Dieser initialisiert dann den *Component Monitor,* um die Ausführung der *Component Steps* zu überwachen. Wie zuvor beschrieben, löst der *Component Monitor* die weiteren Koordinationsbemühungen aus, wenn ein *Component Step* ausgeführt wird, der relevant für die Koordination ist.

### 6.1.4. Zusammenfassung

In diesem Abschnitt wurde eine Middleware präsentiert, die es ermöglicht, verteilte Systeme, die auf dem Konstruktionsparadigma der *Aktiven Komponenten* basieren, mit der Fähigkeit zur Koordination auszustatten, um somit selbstorganisierendes Verhalten realisieren zu können. Konzeptionell basiert die Middleware auf dem Konstruktionsansatz des *Coordination Space* (vgl. Abschnitt 4.3) und der *DeCoMAS* Referenzarchitektur [SR09a]. Dabei wurden die Konzepte aus den Vorarbeiten auf das autonome Verhalten von Aktiven Komponenten und deren serviceorientiertem Kommunikationsparadigma angepasst.

Basierend auf dem den Aktiven Komponenten zugrundeliegenden SCA-Ansatz wurde mit der Middleware ein Konstruktionsansatz realisiert, der Koordination als einen Service für die Komponenten bereitstellt. Aktive Komponenten können dabei spezielle Koordinationsdienste bereitstellen, in denen alle Aspekte, die zur Koordination benötigt werden, gekapselt werden. Die eigentliche Kernfunktionalität der Komponenten bleibt damit unabhängig und lose gekoppelt zur Koordinationsfunktionalität. Somit folgt die Middleware der von [GC92] propagierten strikten Trennung zwischen diesen beiden Aspekten. Der Koordinationsservice stellt dabei eine Schnittstelle bereit, um Koordinationsinformationen zu empfangen und basierend auf diesen Informationen das autonome Verhalten der Komponente zu kontrollieren. Eine weitere Aufgabe des Koordinationsservice ist es, die Ausführung der Aktiven Komponente zu überwachen und den eigentlichen Koordinationsprozess auszulösen, wenn die Komponente einen Zustand erreicht, der für Koordination relevant ist bzw. Koordination erfordert. Der Austausch der für die Koordination erforderlichen Koordinationsinformationen folgt dem serviceorientierten Kommunikationsparadigma der Aktiven Komponenten. Dazu werden spezielle Koordinationsselektoren verwendet, deren Aufgabe es ist, aus der Menge aller verfügbaren Koordinationsservices diejenigen auszuwählen, die für den Koordinationsprozess relevant sind bzw. deren Aktive Komponenten die Koordinationsinformationen erhalten sollen. Die Middleware bietet dabei abstrakte Oberklassen für die Implementierung von Koordinationsselektoren an, die eine entsprechende Methodensignatur zur Auswahl von Koordinationsservices vorgeben und mit dem anwendungsspezifischen Auswahlverfahren erweitert werden können. Neben diesen Auswahlverfahren können die Koordinationsselektoren auch selber Funktionalität besitzen und eigene Berechnungsschritte ausführen. In diesem Fall nehmen sie ähnlich den Koordinationsmedien, die in Abschnitt 4.3.1 beschrieben wurden, eine aktive Rolle ein und sind nicht nur auf die Vermittlung von Nachrichten beschränkt. Nach der Auswahl und der ggf. erfolgten Berechnung werden die selektierten Koordinationsservices aufgerufen, um die Koordinationsinformationen zu erhalten.

Die Beschreibung der Koordinationsprozesse erfolgt in Form einer deklarativen, XML-basierten Beschreibungssprache, die einen Nachfolger der *MASDynamics* Sprache [SR09b] zur Beschreibung von Koordinationspro-

zessen in Multiagentensystemen darstellt. Hier wird in deklarativer Form basierend auf einem Rollenkonzept beschrieben, welche Aktiven Komponenten an den Koordinationsprozessen teilnehmen sollen. Die Koordinationsprozesse werden dabei in der Form beschrieben, dass angegeben wird, welcher Zustand einer aktiven Komponente den Prozess auslösen soll, welche Informationen dabei übertragen werden sollen und welcher Koordinationsselektor zur Auswahl der Empfänger verwendet werden soll. Schließlich wird noch angegeben, welches Verhalten auf Empfängerseite bei Erhalt der Koordinationsinformationen ausgelöst werden soll.

Mit der Middleware wird ein Konstruktionsansatz bereitgestellt, der es erlaubt, verteilte System mit der Fähigkeit zur Koordination auszustatten, indem Koordinationsfunktionalität als Service lose gekoppelt zur eigentlichen Kernfunktionalität von den Komponenten erbracht wird. Der Ansatz folgt dabei Konzepten aus vorangegangenen Arbeiten (*Coordination Space* und *DeCoMAS*) und adaptiert diese für verteilte Systeme im Umfeld der Aktiven Komponenten.

## 6.2. DeCoF: Dezentrales Koordinationsframework

Im bisherigen Verlauf dieser Arbeit wurde mit dem Konstruktionsansatz des *Coordination Space* ein Konzept vorgestellt, dass die Realisierung von selbstorganisierenden Multiagentensystemen basierend auf dezentraler Koordination ermöglicht. Im vorausgegangen Abschnitt wurde eine Middleware präsentiert, welche die Konzepte zur Realisierung von dezentralen Koordinationsprozessen aus dem *Coordination Space* Konstruktionsansatz aufgreift und diese auf verteilte Systeme, die auf dem Modellierungsansatz der *Aktiven Komponenten* basieren, adaptiert. Weiterhin wurde im vorangegangenen Kapitel ein Überblick über adaptive Co-Simulationssysteme gegeben und deutlich gemacht, dass eine essentielle Herausforderung bei der Realisierung solcher Simulationssysteme in der Heterogenität der verwendeten Technologien und Plattformen besteht. Die Lösungen in Form des *Coordination Space* und der im vorherigen Abschnitt beschriebenen Middleware sind aufgrund ihrer Limitierung auf spezielle Unterklassen von verteilten Systemen daher nicht geeignet, um zur Koordination von Co-Simulationssystemen eingesetzt zu werden. Es wird daher ein Konstruktionsansatz benötigt, der es erlaubt, heterogene Systeme unabhängig von der den Systemen zugrundeliegenden Architektur bzw. ihrem Modellierungsansatzes zu koordinieren. In diesem Abschnitt wird daher ein neuartiger Ansatz zur Koordination von heterogenen Systemen präsentiert (vgl. [PDR16a]). Dabei geht es im Folgenden zunächst um zwei der Forschungsfragen, die bereits in der Einleitung zu dieser Dissertation formuliert wurden:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen, um koordinierbar zu sein?

2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

Im Rahmen dieses Abschnitts werden u.a. diese beiden Forschungsfragen beantwortet. Dabei wird zunächst auf das Design und die Architektur eines dezentralen Koordinationsframeworks eingegangen. Darauf aufbauend wird der Ansatz zur Integration des Rahmenwerks in neue oder bereits bestehende Anwendungen präsentiert und es wird ein Überblick über die Beschreibungsmöglichkeiten von Koordinationsprozesse gegeben, ehe schließlich exemplarisch die Verwendung des Rahmenwerks anhand einer Beispielanwendung beschrieben wird.

### 6.2.1. Design und Architektur

Das Design des dezentralen Koordinationsframeworks (DeCoF) basiert auf dem Konzept, dass die selbstorganisierende Dynamik, die dafür sorgt, dass ein System in der Lage ist, sich auf externe und interne Einflüsse hin zu adaptieren, durch dezentrale Koordinationsprozesse abgebildet wird. Diese Prozesse beschreiben das selbstorganisierende Verhalten, das kontinuierlich Aspekte der Anwendung adaptiert, neu strukturiert und reguliert. Dabei instruieren die Prozesse eine Menge von dezentralen Koordinationsmedien und -endpunkten. Die Aufgabe der Koordinationsmedien ist es dabei, sich um die Interaktionen zwischen den Komponenten zu kümmern (Informationsausbreitung), während es die Aufgabe der Endpunkte ist, die Adaption der Komponenten vorzunehmen (lokale Anpassung der Entitäten). Zusammen kontrollieren sie so die mikroskopischen Aktivitäten der Komponenten, die auf makroskopischer Ebene für die Manifestation der gewünschten selbstorganisierenden Dynamik sorgen. Die Integration der Koordinationsmedien und -endpunkte wird dabei durch deklarativ definierte Koordinationsprozesse beschrieben, die sie strukturieren und ihre Operationen instruieren. Der Zusammenhang zwischen der selbstorganisierenden Dynamik und den dezentralen Koordinationsprozessen ist in Abbildung 6.4 dargestellt. Das Konzept orientiert sich dabei stark an dem Ansatz zur Realisierung von selbstorganisierenden Systemen auf Basis von dezentraler Koordination, wie er in der Dissertation von *Sudeikat* [Sud10] erarbeitet wurde (vgl. auch Abbildung 2.6).

Das in [PDR16a] vorgestellte Framework hat sich folglich aus dem *DeCoMAS* [SR09a] und *Coordination Space* [VSL+10] Ansätzen zur softwaretechnischen Verwendung von dezentralen Koordinationsprozessen als wiederverwendbare Designelemente herausgebildet. Konzepte wie die Koordinationsmedien zur Ausbreitung der Koordinationsinformationen und die Koordinationsendpunkte zur Überwachung und Anpassung der lokalen Entitäten wurden ursprünglich im Rahmen des *DeCoMAS* Ansatzes eingeführt. Während die *DeCoMAS* Architektur allerdings speziell dafür entworfen wurde, BDI-Agenten mit der Fähigkeit zur Koordination auszustatten und damit auf

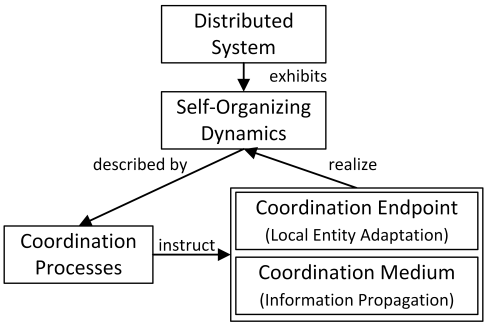



**Abbildung 6.4.:** Selbstorganisierende Dynamik eines verteilten Systems basierend auf dezentralen Koordinationsprozessen - nach (PDR16a).

diese Klasse von verteilten Anwendungen beschränkt ist, zielt der *DeCoF* Ansatz darauf ab, verteilte Systeme im Allgemeinen zu unterstützen.

Die konzeptionelle Architektur des vorgestellten Frameworks ist in Abbildung 6.5 dargestellt. Komponenten, die mit der Fähigkeit zur Koordination ausgestattet werden sollen, werden als *Coordinatable Components* bezeichnet. Da das Framework darauf abzielt, unterschiedliche Arten von Systemen zu unterstützen und sich auf keine spezifische Systemarchitektur beschränkt, gibt es keine inhärenten Charakteristika der Komponenten, die verwendet werden können, um ihr Verhalten zu observieren bzw. zu beeinflussen. Unterschiedliche Arten von Multiagentensysteme verwenden unterschiedliche Konzepte zum Scheduling und um den Lebenszyklus ihrer Komponenten zu verwalten. Komponentenbasierte Systeme bieten unter Umständen keine solchen Konzepte an. Daher wurde in [PDR16a] das Konzept der *Coordination Events* eingeführt. Dieses sind Ereignisse, die von einer koordinierbaren Komponente ausgelöst werden, sobald ein für die Koordination relevantes Ereignis innerhalb der koordinierbaren Komponente auftritt. Ein Koordinationsereignis $ce$ ist dabei als ein 2-Tupel definiert, das kontextualisierte Daten über das spezifische Koordinationsereignis $cd$ sowie eine Repräsentation des Verursachers des Ereignisses $eo$ enthält. Ein solches Koordinationsereignis ist daher definiert als: $ce := (cd, eo)$.

Basierend auf einer strikten Trennung zwischen der Anwendungs- und der Koordinationslogik, wie sie von [GC92] propagiert wurde, wird die eigentliche Verarbeitung von Koordinationsereignissen von den entsprechenden Koordinationsendpunkten (*Coordination Endpoint*) (vgl. Abbildung 6.4) übernommen. Die Endpunkte sind dabei lose an die koordinierbaren Komponenten durch einen sogenannten *Coordination Event Bus* gekoppelt. Ein solcher *Eventbus*[1] ermöglicht es, lose gekoppelte *publish-subscribe* Kommunikationsmuster zwischen zwei Komponenten zu realisieren, ohne das die Komponenten dabei explizit voneinander wissen müssen. Somit ist es möglich, dass die Komponenten alleine für die Umsetzung der Anwendungslogik verantwortlich sind und

[1] `https://github.com/google/guava/wiki/EventBusExplained`, aufgerufen am 1. Februar 2017

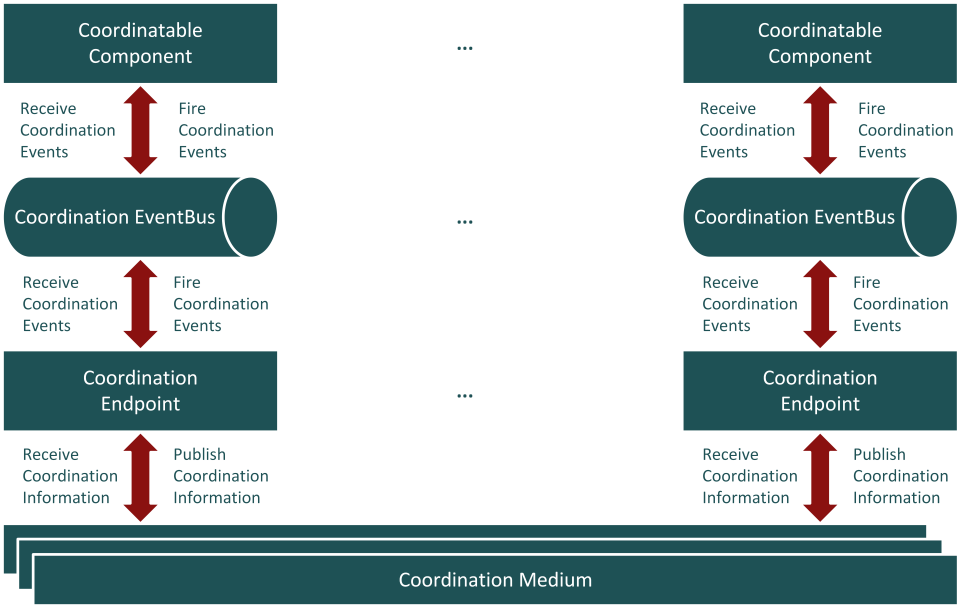



**Abbildung 6.5.:** Architektur des dezentralen Koordinationsframework - nach (PDR16a).

kein Wissen über das mögliche Vorhandensein eines Koordinationsendpunktes benötigten.

Immer, wenn eine koordinierbare Komponente ein Koordinationsereignis auslöst, wird dieses vom zugehörigen Koordinationsendpunkt über den Koordinationsbus empfangen. Der Endpunkt verarbeitet dieses Ereignis entsprechend der vordefinierten Koordinationsprozessbeschreibung. Die Prozessbeschreibung definiert dabei, wie unterschiedliche Koordinationsereignisse von den Endpunkten verarbeiten werden sollen. Diese Beschreibungen beinhalten dabei Informationen darüber, *wie* das Koordinationsereignis verteilt werden soll und *welche* anderen koordinierbaren Komponenten es erhalten sollen. Das *wie* ist beschrieben, indem angegeben wird, welche Art von Koordinationsmedium für die Informationsausbreitung verwendet werden soll. Wie bereits beschrieben wurde, sind die Koordinationsmedien für die Informationsausbreitung unter den Komponenten verantwortlich (vgl. Abbildung 6.4).

Die Auswahl der Empfänger ist über ein Rollenkonzept realisiert. Dabei kann eine Komponente mehrere Rollen haben und eine Rolle aus mehreren unterschiedlichen Komponenten bestehen. Um ein Koordinationsereignis zu verarbeiten, kapselt ein Endpunkt es und reichert es mit zusätzlichen Informationen über den Ursprungsendpunkt an. Die daraus resultierende Koordinationsinformation $ci$ ist ein wieder ein 2-Tupel, welches das Koordinationsereignis $ce$ und Informationen über den Ursprungsendpunkt $oe$ enthält. Es ist daher definiert als: $ci := (ce, oe)$. Zusätzlich zur Beschreibung darüber, welches Koordinationsereignis, das von welcher koordinierbaren Komponente ausgeht, zu welchen anderen koordinierbaren Komponenten publiziert werden soll, enthält eine Koordinationsprozessbeschreibung noch die Information, welches Koordinationsereignis daraufhin in den empfangenden Komponenten ausgelöst wer-

den soll. Wie die Koordinationsinformationen dabei tatsächlich unter den Komponenten propagiert werden, ist Teil der Implementierung eines Koordinationsmediums. Das betrifft sowohl die technische Realisierung der Informationsverteilung als auch die fachliche Auswahl einer möglichen Untermenge von Empfängern. Daher sind einfache Koordinationsmedien, die direkt von der Netzwerktopologie abhängen, ebenso denkbar wie komplexere Ausprägungen, bei denen die Ausbreitung der Informationen z.B. von Diffusionsprozessen in einer (virtuellen) Umgebung abhängt. Zusätzlich ist es auch hier möglich, dass die Koordinationsmedien eigene Berechnungsschritte durchführen, also ein aktives Verhalten zeigen.

### 6.2.2. Integrationsansatz

Grundsätzlich kann zwischen zwei unterschiedlichen Arten von Software-Komponenten unterschieden werden: *Managed-* und *Unmanaged-Components*. Bei *Managed-Components* kann es sich z.B. um Softwareagenten oder auch Aktive Komponenten handeln. Solche Komponenten haben gemeinsam, dass es spezifische Elemente gibt, mit denen ihr Verhalten zur Designzeit beschrieben werden kann. Zur Laufzeit werden solche Komponenten in einer bestimmten Ausführungsumgebung wie z.B. der *Jadex Active Components* Plattform ausgeführt. Diese Ausführungsumgebungen stellen dabei bestimmte Mechanismen zur Verfügung, um den *Lifecycle* und das *Scheduling* der Komponenten zu steuern. Weiterhin bieten die Ausführungsumgebungen i.d.R. Möglichkeiten, um den Zustand oder die Ausführung der *Managed-Components* zu überwachen. All dieses sind Mechanismen, die von einer Koordinationsinfrastruktur verwendet werden können, um die Komponenten zu koordinieren. Für die Koordination von besonderer Bedeutung ist dabei die Fähigkeit, den Zustand bzw. die Ausführung der Komponenten überwachen zu können, um Fälle zu erkennen, in denen Koordination erforderlich ist. Gleichermaßen entscheidend für die erfolgreiche Integration von Koordinationsfunktionalität ist es, die Ausführung der Komponenten beeinflussen zu können, um im Rahmen der Koordinationsbemühungen die Komponenten steuern zu können.

Im Fall von *Unmanaged-Components* stehen solche Möglichkeiten nicht zur Verfügung. Hierbei gibt es keine impliziten Elemente zur Beschreibung von Verhalten zur Designzeit. Da solche Komponenten i.d.R. nicht in einer bestimmten Ausführungsumgebung situiert sind, stehen auch keine Möglichkeiten zum Steuern des *Lifecycles* und des *Schedulings* bereit. Außerdem gibt es keine impliziten Möglichkeiten, das Verhalten oder den Zustand der Komponenten zu überwachen. Hat man die Zielsetzung der Koordination von heterogenen, verteilten Systemen, wie sie z.B. im Umfeld der Co-Simulationen vorkommen, vor Augen, muss ein möglichst allgemein gültiger Koordinationsansatz entwickelt werden. Dieser Ansatz sollte nicht nur die Koordination von *Managed-Components*, sondern muss auch die von *Unmanaged-Components* unterstützten. Da solche nicht verwalteten Kom-

ponenten über keine impliziten Möglichkeiten der Überwachung und Einflussnahme verfügen, wurde in [PDR16a] die Verwendung expliziter Koordinationsereignisse vorgeschlagen, um somit auch *Unmanaged-Components* koordinierbar zu machen.

Die Verwendung expliziter Koordinationsereignisse bedingt dabei, dass die Komponenten, die mit der Fähigkeit zur Koordination ausgestattet werden sollen, das `ICoordinatable` Interface implementieren müssen. Diese Schnittstelle gibt an, dass die Komponenten zwei Methoden implementieren müssen (siehe auch Abbildung 6.6):

- Die `getId` Funktion gibt dabei einen Zeichenstring wieder, mit dem die Komponente eindeutig im Kontext der verteilten Anwendung identifiziert werden kann. Eine solche Funktionalität wird benötigt, wenn eine Komponente im Rahmen der Koordination direkt adressiert werden soll und um den Erzeuger eines Koordinationsereignisses identifizieren und benennen zu können.
- Die `handleCoordinationEvent` Methode wird von den zugeordneten Koordinationsendpunkten auf den zu koordinierenden Komponenten aufgerufen, wenn diese über ein Koordinationsmedium ein Koordinationsereignis für die Komponenten empfangen. An dieser Stelle muss die komponentenspezifische Handhabung der Koordinationsereignisse realisiert werden, die spezifiziert, wie sich eine Komponente verhält, wenn sie ein Koordinationsereignis erhält.

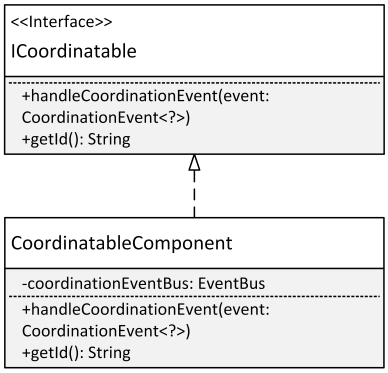



**Abbildung 6.6.:** UML-Klassendiagramm zur Schnittstelle für koordinierbare Komponenten.

Die Handhabung der Koordinationsereignisse in den Komponenten hängt dabei stark von der zugrundeliegenden Architektur der Komponenten ab. Handelt es sich bei den Komponenten z.B. um Softwareagenten oder Aktive Komponenten, deren Ausführung von einer entsprechenden Ausführungsumgebung gesteuert wird, dann muss in der Methode die Verarbeitung des Koordinationsereignisses im Rahmen der plattformspezifischen Scheduling Mechanismen erfolgen. Da diese Funktionalität plattformabhängig ist, kann vom *DeCoF* keine fertige Lösung bereitgestellt, sondern nur der abstrakte Rahmen für die tatsächliche Ausimplementierung vorgegeben werden. An dieser

Stelle unterscheidet sich die *DeCoF* Lösung mit ihrem Framework-Charakter auch von Lösungen wie dem *Coordination Space* oder *DeCoMAS*, die fertige Lösungen für jeweils spezifische Ausführungsumgebungen anbieten. Unter einem Framework wird in der Regel ein Rahmen verstanden, der für eine bestimmte Gruppe von Programmen oder Softwareentwicklungswerkzeugen ein Skelett vorgibt, das um eigene Programme ergänzt werden kann. Frameworks treten häufig in Form von Klassenhierarchien oder Klassenbibliotheken auf, die (abstrakte) Klassen und Schnittstellen enthalten, die in der Regel unter Verwendung einer einzigen Programmiersprache kodiert wurden und Gegenstand einer White-Box-Verwendung sind (vgl. [HGRZ06]).

Die Verwendung expliziter Koordinationsereignisse zur Unterstützung von *Unmanaged-Components* bedingt dabei, dass koordinierbare Komponenten explizit solche Ereignisse auslösen. Dadurch wird die strikte Trennung zwischen Anwendungs- und Koordinationslogik, wie sie von [GC92] propagiert wurde, etwas aufgeweicht, da nun explizit in den Komponenten das Auslösen solcher Ereignisse realisiert werden muss. Andernfalls wäre es nicht möglich, die Ausführung der Komponenten in Hinblick auf für die Koordination relevante Ereignisse zu überwachen, da *Unmanaged-Components* im Gegensatz zu ihren verwalteten Gegenstücken die Observierungsmöglichkeiten der Ausführungsumgebung fehlen. Durch die Verwendung des zuvor beschriebenen Koordinationsbusses benötigen die Komponenten allerdings kein explizites Wissen darüber, ob ihr Zustand von einem Endpunkt zu Koordinationszwecken überwacht wird. Somit kann eine lose Kopplung zwischen der Komponente und ihrem Endpunkt realisiert werden, bei der die Komponente zwar explizit Koordinationsereignisse auslösen muss, aber kein explizites Wissen über die Anwesenheit bzw. Abwesenheit eines Endpunktes benötigt.

Das im Rahmen dieser Dissertation entwickelte dezentrale Koordinationsframework, das in [PDR16a] präsentiert wurde, liefert einen Implementationsrahmen für verteile Anwendungen, die in der Programmiersprache *Java* realisiert sind. Im weiteren Verlauf der Integration des Frameworks muss nach Implementierung des `ICoordinatable` Interfaces für die zu koordinierenden Komponenten ein Koordinationsendpunkt (`CoordinationEndpoint`) bereitgestellt werden. Das Framework bietet hierfür eine statische Hilfsmethode an, die den Endpunkt erstellt und die Komponente über das zuvor beschriebene Konzept des Koordinationsbusses (`EventBus`) lose gekoppelt mit der Komponente verbindet. Das Framework bietet dabei eine fertige Implementierung des Koordinationsendpunkts sowie eine generische Implementation, um Koordinationsinformationen beliebigen Typs zu kapseln (`CoordinationInformation<T>`). Zur anwendungsspezifischen Implementierung der Koordinationsmedien und -ereignisse stehen abstrakte Oberklassen zur Verfügung (`CoordinationMedium` und `CoordinationEvent<T>`). Abbildung 6.7 zeigt in Form eines UML Klassendiagramms, wie die wichtigsten Klassen des dezentralen Koordinationsframeworks zusammenhängen. Die Methodologie, um eine verteilte Anwendung mit dezentralen Koordinationsprozessen auszustatten, umfasst zusammenfassend folgende Schritte:

1. Schreiben der XML-basierten deklarativen Koordinationsprozessbeschreibungen, die das Verhalten der Koordinationsendpunkte instruieren. Eine detaillierte Beschreibung der entsprechenden Beschreibungssprache erfolgt im nächsten Abschnitt.

2. Implementierung des `ICoordinatable` Interfaces für die Komponenten, die mit der Fähigkeit zur Koordination ausgestattet werden sollen.

3. Identifikation der Ereignisse, die für die Koordination relevant sind und deren explizite Implementierung als Koordinationsereignisse durch Verwendung der abstrakten `CoordinationEvent<T>` Oberklasse.

4. Implementierung der Koordinationslogik zur anwendungsspezifischen Ausbreitung der Koordinationsinformationen durch Verwendung der abstrakten `CoordinationMedium` Oberklasse.

Ein konkretes Beispiel für die Verwendung des DeCoF zur dezentralen Koordination in einer verteilten Anwendung wird in Unterabschnitt 6.2.4 gegeben.

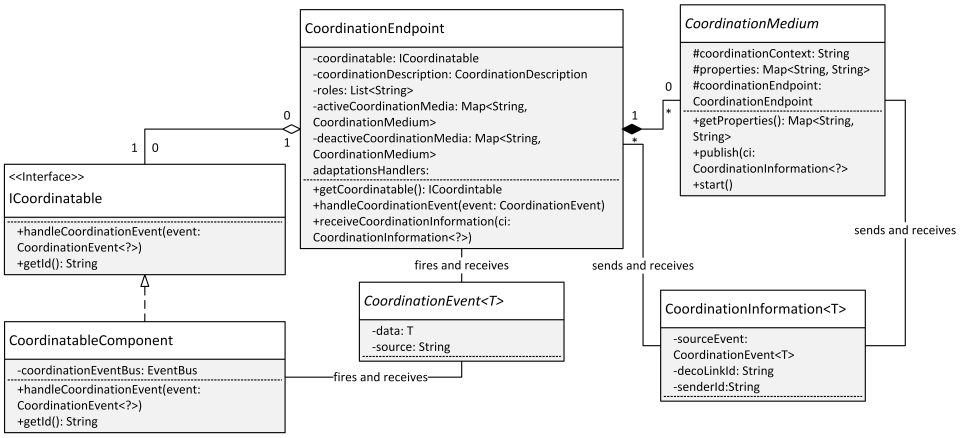



**Abbildung 6.7.:** UML Klassendiagramm des dezentralen Koordinationsframeworks mit den wichtigsten Klassen bzgl. der Koordination - nach (PDR16a).

### 6.2.3. Beschreibung von Koordinationsprozessen

Ein Koordinationsprozess, wie er in der Dissertation von *Sudeikat* [Sud10] definiert und im Rahmen dieser Arbeit verwendet wird, setzt sich grundsätzlich aus drei unterschiedlichen Typen von Elementen zusammen. Zunächst wird in Form eines Rollenkonzepts beschrieben, welche Entitäten in welcher Form am Koordinationsprozess beteiligt sind. Die Rollen abstrahieren dabei von den Implementierungsklassen der Komponenten und erlauben es, gemeinsames Verhalten im Bezug auf die Koordination für Komponenten unterschiedlichen Typs zu definieren. Zwischen den Rollen und Komponenten besteht eine $m : n$ Beziehung, das bedeutet, dass Rollen Komponenten unterschiedlichen Typs enthalten können und eine Komponente unterschiedlichen Rollen zugeordnet sein kann. Weitere für die Koordinationsprozesse wichtige

Elemente sind solche, die die Zustände bzw. das Verhalten der betroffenen Entitäten abbilden. *DeCoF* verwendet an dieser Stelle das Konzept der explizit modellierten Koordinationsereignisse. Der letzte Elementtyp, der zur Beschreibung eines Koordinationsprozesses benötigt wird, ist das Koordinationsmedium. Mit dem Medium wird die technische Realisierung der Informationsausbreitung angegeben, die bestimmt, wie Informationen im Rahmen des Koordinationsprozesses unter den teilnehmenden Komponenten verteilt werden sollen. Abbildung 6.8 setzt all diese Elemente in einem gemeinsamen Kontext und zeigt dabei grafisch den Aufbau bzw. die Zusammensetzung eines Koordinationsprozesses nach [Sud10], der mit DeCoF realisiert werden kann. Basierend auf dem Rollenkonzept wird mit dem Koordinationsprozess beschrieben, welcher Typ eines Koordinationsereignisses, der in einer Komponente mit der entsprechenden Rolle auftritt, auf welchen anderen Typ von Koordinationsereignis in welcher anderen Komponente abgebildet werden soll und welches Koordinationsmedium dabei verwendet werden soll, um die Informationen zu verteilen.

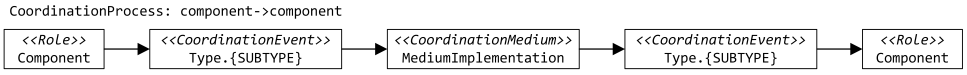



**Abbildung 6.8.:** Visuelle Repräsentation eines generischen Koordinationsprozesses - aus (PDR16a, S. 7).

Zur deklarativen Beschreibung der Koordinationsprozesse wurde in Anlehnung an die *MASDynamics* Sprache [SR09b] eine XML-basierte Koordinationssprache entwickelt. Abbildung 6.9 stellt ein UML Klassendiagramm dar, das die Elemente dieser Beschreibungssprache in einen grafischen Kontext setzt. Zu einer Koordinationsbeschreibung (`CoordinationDescription`) gehört dabei zunächst ein Kontext (`context`), dieser ist beschreibender Natur und gibt der verteilten Anwendung einen Namen. Des Weiteren besteht eine Koordinationsbeschreibung aus einer Reihe von Rollendefinitionen (`roleDefinitions`). Eine Rolle (`Role`) hat dabei einen Namen (`name`), unter dem sie in der Koordinationsbeschreibung referenziert wird, und beinhaltet eine Liste von Komponenten (`components`), die dieser Rolle zugeordnet sind. Eine Komponente (`Component`) wird in der Beschreibungssprache referenziert, indem dort der Name der Klasse (`clazz`) angegeben wird, welche die Komponente implementiert.

Die eigentlichen Koordinationsprozesse sind in Form sogenannter dezentraler Koordinationsverbindungen (`DecoLink`) beschrieben. Dabei kann zunächst angegeben werden, ob die Koordination entkoppelt vom eigentlichen Ausführungsthread der Komponenten stattfinden soll (`decoupled`). Wenn dieses gewünscht ist, erzeugt der Koordinationsendpunkt der Komponenten einen eigenständigen Thread, in dem die Koordination komplett nebenläufig und entkoppelt von der Ausführungsumgebung der Komponente stattfindet. Ein solches Verhalten ist i.d.R. gewünscht, wenn es sich um *Unmanaged-Components* handelt, deren Ausführung nicht verwaltet wird, um dort die

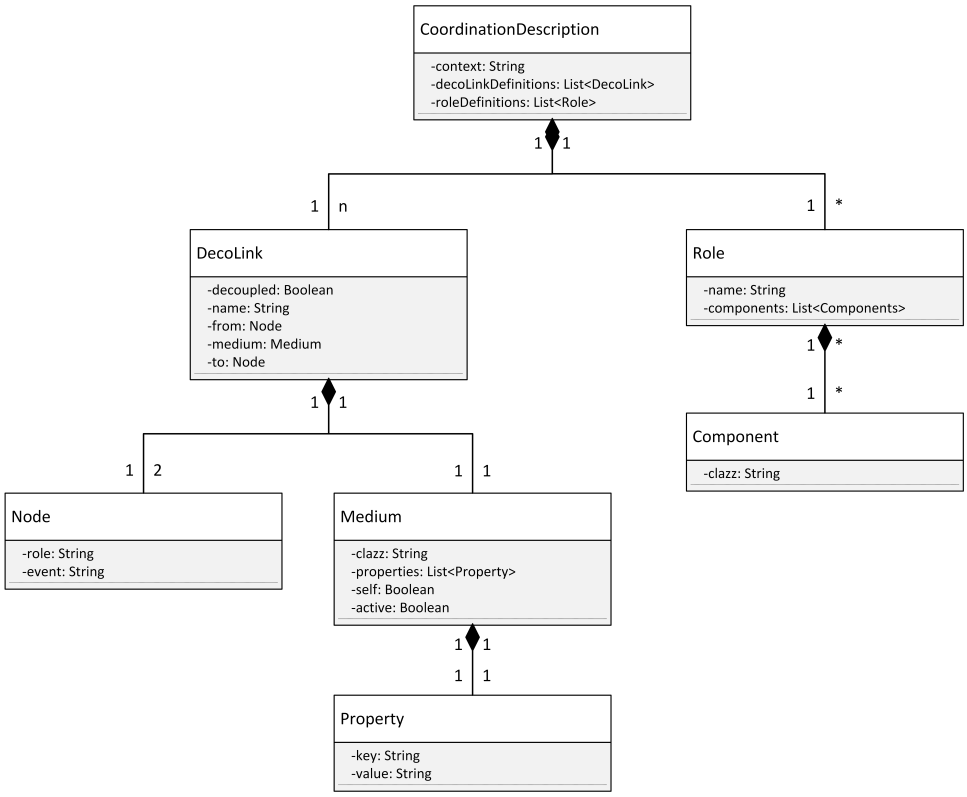



**Abbildung 6.9.:** UML Klassendiagramm der DeCoF Koordinationsbeschreibungssprache.

Berechnungen, die zur Koordination benötigt werden, entkoppelt und damit parallel zum Anwendungsthread durchzuführen. Im Falle von *Managed-Components* kann es zu Fehlern führen, wenn die Ausführungsumgebung der Komponenten umgangen wird und externe Threads gestartet werden. Um solche möglichen Interferenzen zu vermeiden, bietet *DeCoF* daher an dieser Stelle die Möglichkeit, das gewünschte Verhalten explizit zu konfigurieren. Zu einem `DecoLink` gehören des Weiteren ein Name (`name`), der beschreibende Eigenschaften hat. Die eigentliche Konfiguration der Verbindung zwischen den beteiligten Rollen erfolgt über die Angabe von Absender- (`from`) und Empfängerknoten (`to`) der Koordinationsereignisse. Ein Knoten (`Node`) gibt dabei an, welches Koordinationsereignis (`event`) in welcher Rolle (`role`) auf Absenderseite den Koordinationsprozess auslösen soll bzw. welches Koordinationsereignis in welcher Rolle auf Empfängerseite daraufhin ausgelöst werden soll. Schließlich wird im Rahmen des `DecoLink` noch angegeben, welches Koordinationsmedium (`medium`) verwendet werden soll.

Für das Koordinationsmedium (`Medium`) muss an dieser Stelle zunächst der Name der Implementationsklasse des Mediums angegeben werden (`clazz`). Zusätzlich verfügt das Medium noch über eine Reihe von Konfigurationsmöglichkeiten. Diese können in Form einer Eigenschaftenliste (`properties`) angegeben werden. Eine solche Eigenschaft (`Property`) ist dabei ein einfaches zeichenbasiertes *Key-Value*-Paar. Zusätzlich zu diesen ge-

nerischen Konfigurationsmöglichkeiten kann über ein Flag gesteuert werden, ob ein Medium Koordinationsinformationen grundsätzlich auch an den Absender verteilen soll oder nicht (`self`). Außerdem kann das Medium zur Startzeit über ein weiteres Flag aktiv oder inaktiv gesetzt werden (`active`). Zur Startzeit deaktivierte Koordinationsmedien spielen im nächsten Abschnitt eine Rolle, wenn es um die strukturelle Adaption von Koordinationsprozessen in *DeCoF* geht. Zunächst wird aber im nächsten Unterabschnitt ein Beispiel zur Integration von Koordination in ein Multiagentensystem auf Basis von *DeCoF* präsentiert. In diesem Rahmen wird auch ein Beispiel für eine deklarative Koordinationsprozessbeschreibung gegeben.

### 6.2.4. Integrationsbeispiel: Jade Party

Der Hauptgrund für die Entwicklung des *DeCoF* war es, ein Koordinationsframework zur Verfügung zu stellen, um unterschiedliche Arten von verteilten Systemen koordinieren zu können. In diesem Unterabschnitt wird ein Beispiel präsentiert, das die Verwendung im Zusammenspiel mit dem weit verbreiteten *Java Agent DEvelopment Framework* (JADE) [BPR99] aufzeigt. Das Beispiel greift dabei das Szenario der Jade Party[2] auf, bei der ein Gastgeber (Host) eine Party ausrichtet, zu der er eine Anzahl von Gästen einlädt. Das Szenario ist dabei wie folgt aufgebaut: Sobald die Party startet, erstellt der Gastgeber $N$ Gast-Agenten. Diese Gäste melden ihre Ankunft dem Gastgeber, indem sie eine *hello* Nachricht an diesen schicken. Der Gastgeber wählt zufällig einen Gast aus und erzählt diesem ein Gerücht (*rumor*). Dann wählt der Gastgeber zufällig zwei Gäste aus und macht diese miteinander bekannt (*introduce*). Die Party läuft nun wie folgt ab: Jeder Gast, der einem anderen vorgestellt wurde, bittet den Gastgeber, ihm auch noch einen weiteren zufälligen Gast vorzustellen. Wenn ein Gast jemanden vorgestellt wird, teilt er diesem, sofern er es schon kennt, das Gerücht mit. Wenn ein Gast das Gerücht zum ersten Mal hört, teilt er dieses dem Gastgeber mit. Sobald alle Gäste das Gerücht gehört haben, endet die Feier.

Im ursprünglichem Beispiel wurden FIPA[3] Nachrichten verwendet, um die Interaktionen zwischen den Gästen in Nachrichtenform zu realisieren. Um nun die Verwendung des *DeCoF* in Verbindung mit *JADE* zu demonstrieren, wurde die Nachrichten-basierte Interaktion durch drei Koordinationsprozesse ersetzt. Diese Prozesse bilden die Interaktionen zwischen einem Gast und dem Gastgeber (1), zwischen dem Gastgeber und einem Gast (2) und zwischen zwei Gästen ab (3). Dazu wurden zwei Rollen eingeführt, die des Gastgebers und die eines Gastes. Außerdem musste das `ICoordinatable` Interface für beiden Agententypen implementiert und diese mit einem Koordinationsendpunkt ausgestattet werden. Abbildung 6.10 zeigt eine visuelle Repräsentation der drei Koordinationsprozesse. Wenn ein Gast einem anderen

[2] `http://jade.tilab.com/documentation/examples/party/`, aufgerufen am 1. Februar 2017

[3] `http://www.fipa.org/`, aufgerufen am 1. Februar 2017

Gast vorgestellt werden möchte, bei der Party ankommt oder das Gerücht zum ersten Mal hört, löst er ein entsprechendes Koordinationsereignis aus. Dieses Ereignis wird vom Koordinationsendpunkt des Agenten verarbeitet und das in der Koordinationsbeschreibung referenzierte Koordinationsmedium verwendet, um die Koordinationsinformation zu verteilen. Entsprechend für den zweiten Koordinationsprozess: Wenn der Gastgeber einen Gast einem anderen Gast vorstellt, einem Gast das Gerücht erzählt oder einen Gast auffordert, die Feier zu verlassen, wird ein entsprechendes Koordinationsereignis vom Gastgeber-Agenten ausgelöst. Der dritte Koordinationsprozess bildet schließlich die Interaktionen zwischen den Gästen ab: Ein Gast begrüßt einen anderen Gast oder erzählt diesem das Gerücht.

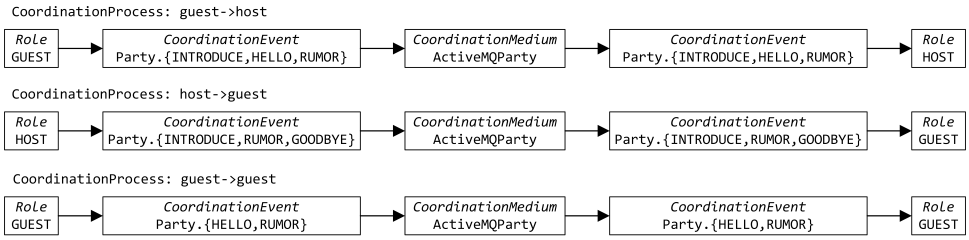



**Abbildung 6.10.:** Visuelle Repräsentation der Koordinationsprozesse des Jade Party Beispiels - aus (PDR16a, S. 10).

In diesem Beispiel verwenden alle drei Koordinationsprozesse ein Koordinationsmedium, das den *Apache ActiveMQ*[4] Message Broker verwendet. Dadurch entsteht eine Infrastruktur, bei der eine Message Queue (Nachrichtenschlange) für jeden Koordinationsendpunkt erstellt wird. Die Koordinationsinformationen werden verteilt, indem sie in die Message Queue des Empfängers publiziert werden. Das Beispiel soll dabei verdeutlichen, wie unterschiedliche Technologien mit *DeCoF* kombiniert werden können, um so verteilte, heterogene und skalierbare Anwendungen zu realisieren. Wie bereits zuvor beschrieben, entpackt ein Koordinationsendpunkt das umschlossene Koordinationsereignis, sobald er die entsprechenden Koordinationsinformationen über ein Koordinationsmedium erhält und publiziert das Ereignis über die `handleCoordinationEvent` Methode im `ICoordinatable` Interface, an die entsprechende Komponenten. Wie die Komponente, in diesem Fall ein *JADE* Agent, das Ereignis verarbeitet, muss entsprechend plattformspezifisch in der Methode implementiert werden. Wenn, wie in diesem Fall, ein Agentenframework verwendet wird, das eigene Lifecycle und Scheduling Mechanismen hat, muss die Verarbeitung des Koordinationsereignisses entsprechend in diese integriert werden. Im Fall dieses Beispiels geschieht dieses, indem innerhalb der `handleCoordinationEvent` Methode ein *JADE*-spezifisches `OneShotBehavior` dem Agenten hinzugefügt wird, sodass die *JADE*-Plattform das Verhalten entsprechend planen und ausführen kann. In Quelltext 6.1 ist die deklarative, XML-basierte Beschreibung der drei beschriebenen Koordinationsprozesse aus diesem Beispiel aufgeführt.

[4] `http://activemq.apache.org/`, aufgerufen am 1. Februar 2017

```xml
<?xml version="1.0" encoding="UTF-8"?>
<coordination-description context="PartyTime">
 <role-definitions>
  <role name="host">
   <components>
    <component class="de.haw.c4das.decof.examples.decoparty.HostAgent" />
   </components>
  </role>
  <role name="guest">
   <components>
    <component class="de.haw.c4das.decof.examples.decoparty.GuestAgent" />
   </components>
  </role>
 </role-definitions>
 <deco-link-definitions>
  <!-- Guest->Host Link -->
  <deco-link id="guest-host">
   <from role="guest" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
   <medium
    class="de.haw.c4das.decof.examples.decoparty.ActiveMQPartyCoordinationMedium"
    self="false">
    <properties>
     <property key="brokerURL" value="tcp://localhost:61616/" />
    </properties>
   </medium>
   <to role="host" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
  </deco-link>

  <!-- Host->Guest Link -->
  <deco-link id="host-guest">
   <from role="host" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
   <medium
    class="de.haw.c4das.decof.examples.decoparty.ActiveMQPartyCoordinationMedium"
    self="false">
    <properties>
     <property key="brokerURL" value="tcp://localhost:61616/" />
    </properties>
   </medium>
   <to role="guest" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
  </deco-link>

  <!-- Guest->Guest Link -->
  <deco-link id="guest-guest">
   <from role="guest" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
   <medium
    class="de.haw.c4das.decof.examples.decoparty.ActiveMQPartyCoordinationMedium"
    self="false">
    <properties>
     <property key="brokerURL" value="tcp://localhost:61616/" />
    </properties>
   </medium>
   <to role="guest" event="de.haw.c4das.decof.examples.decoparty.PartyEvent" />
  </deco-link>
 </deco-link-definitions>
</coordination-description>
```

**Quelltext 6.1:** Deklarative XML Beschreibung der drei Koordinationsprozesse aus dem *Jade* Party Beispiel.

### 6.2.5. Zusammenfassung

In diesem Abschnitt wurde ein dezentrales Koordinationsframework präsentiert, das im Rahmen dieser Dissertation in [PDR16a] erarbeitet wurde.

Das präsentierte Framework zielt dabei darauf ab, einen Konstruktionsansatz zu liefern, der es erlaubt, heterogene Systeme unabhängig von der den Systemen zugrundeliegenden Architektur zu koordinieren. Dabei hat der Ansatz besonders die Anforderungen aus dem Umfeld der Co-Simulationen bzgl. der Heterogenität von Systemarchitekturen im Fokus. Im vorangegangen Verlauf dieser Arbeit wurden mit dem *DeCoMAS* Ansatz [SR09a] zur Koordination von Multiagentensystemen und dem *Coordination Space* Konstruktionsansatz [VSL+10], der eine Weiterentwicklung der *DeCoMAS* Konzepte darstellt, bereits etablierte Ansätze präsentiert, die sich bei der Koordination von Multiagentensystemen bewährt haben. Aber genauso wie die im vorherigen Abschnitt präsentierte Koordinationsmiddleware (siehe Abschnitt 6.1) zielen diese Ansätze auf eine bestimmte Form von Systemarchitekturen ab und ermöglichen es nicht, verteilte Systeme im Allgemeinen zu koordinieren. Das in diesem Abschnitt präsentierte dezentrale Koordinationsframework (DeCoF) liefert mit seinem neuartigen Ansatz zur Koordination von heterogenen Systemen für solche Fälle eine Lösung. Damit ermöglicht es der Ansatz auch die Integrationsarchitekturen für Co-Simulationssysteme umzusetzen, die im vorangegangenen Kapitel präsentiert wurden. Grundsätzlich wird es somit möglich, die Aktivitäten von Komponenten in Co-Simulationssystemen zu koordinieren, sodass die verteilte Ausführung eines solchen Systems auch auf Basis von dezentralen Koordinationsprozessen gesteuert werden kann.

In der Einleitung zu diesem Abschnitt wurden die folgenden zwei Forschungsfragen aufgegriffen:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen um koordinierbar zu sein?

2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

Bezugnehmend auf die erste Frage lässt sich dabei feststellen, dass es einen grundsätzlichen Unterschied zwischen der Koordination von *Managed-* und *Unmanaged-Components* gibt. Während Erstere, z.B. Agenten in einem Multiagentensystem, von einer Ausführungsumgebung verwaltet werden, die zur Entwicklungszeit spezielle Modellierungselemente zur Verfügung stellt, welche zur Ausführungszeit observiert werden können, um Zustände oder Ereignisse zu erkennen, die für die Koordination relevant sind und daher einen Koordinationsprozess auslösen sollen, fehlt es bei *Unmanaged-Components* an solchen Elementen. Daher müssen zur Koordination von generischen Software-Komponenten explizite Koordinationsereignisse verwendet werden, die von den entsprechenden Komponenten ausgelöst werden, wenn ein Zustand, der für die Koordination relevant ist, erreicht wurde. Dadurch wird die strikte Trennung zwischen Koordinations- und Anwendungslogik, wie sie u.a. von [GC92] propagiert wird, etwas aufgeweicht. Mit dem Konzept des Koordinationsbusses wurde im Rahmen von *DeCoF* allerdings ein Ansatz beschrieben,

bei dem zwar das Auslösen von Koordinationsereignissen explizit in den Komponenten geschehen muss, allerdings erlaubt der Koordinationsbus eine lose Kopplung zwischen den Komponenten und den Koordinationsendpunkten, welche die Koordinationsfunktionalität für die Komponenten kapseln und bereit stellen. Die Komponenten brauchen kein explizites Wissen darüber, ob sie von einem Koordinationsendpunkt überwacht und ggf. beeinflusst werden. Die Anwendungslogik in den Komponenten kann somit unabhängig von der Koordinationslogik realisiert werden und muss nur an geeigneten Stellen ein Koordinationsereignis auslösen, sich aber nicht weiter um dessen Verarbeitung kümmern.

Die zweite Forschungsfrage wurde in diesem Abschnitt in Form der präsentierten *DeCoF*-Architektur beantwortet (siehe Unterabschnitt 6.2.1). Die Architektur zeigt dabei auf konzeptioneller Ebene, wie die Koordination von generischen Softwarekomponenten auf Basis der Konzepte der Koordinationereignisse, des Koordinationbus, der Koordinationsendpunkte und der Koordinationsmedien erfolgen kann. Auf technischer Ebene wurde in Unterabschnitt 6.2.2 der Integrationsansatz des präsentierten Rahmenwerks vorgestellt. Dabei wurde mit dem `ICoordinatable` Interface die Schnittstelle präsentiert, die Komponenten implementieren müssen, wenn sie mit der Fähigkeit zur Koordination ausgestattet werden sollen. Dabei wurde auf technischer Ebene auch die Verwendung bzw. Handhabung von Koordinationsereignissen durch die Komponenten beschrieben und ein Überblick über die wichtigsten Klassen gegeben, die bei der Verwendung des Rahmenwerks eine Rolle spielen. Wie schließlich konkret Koordinationsprozesse mit *DeCoF* modelliert werden können, wurde in Unterabschnitt 6.2.3 beschrieben. Hierbei wurde sowohl der konzeptuelle Ansatz der Koordinationsprozesse im Kontext von *DeCoF* beschrieben als auch ein Überblick über die Sprachelemente der deklarativen Beschreibungssprache gegeben. Schließlich wurde in Unterabschnitt 6.2.4 mit der *Jade Party* ein Beispiel präsentiert, in dem die Verwendung von *DeCoF* im Zusammenspiel mit der weitverbreiteten Agentenplattform *JADE* aufgezeigt wurde. Dabei wurde insbesondere darauf eingegangen, wie die `ICoordinatable` Schnittstelle im Zusammenhang mit plattformspezifischen Scheduling-Mechanismen verwendet werden kann und wie im Rahmen von *DeCoF* unterschiedliche Technologien zur Realisierung von verteilten Systemen kombiniert werden können, im Fall des Beispiels die *JADE*-Agentenplattform und der *Apache ActiveMQ* Message Broker.

## 6.3. Strukturelle Adaption von Koordinationsprozessen

In Abschnitt 4.4 wurde bereits das Konzept zur strukturellen Adaption von Koordinationsprozessen beschrieben, bei dem es um den dynamischen Austausch der Koordinationsprozesse bzw. deren Rekonfiguration zur Laufzeit geht, um unerwünschtes Verhalten in selbstorganisierenden Systemen zu überwinden (siehe auch [PRD15]). Da die Koordinationsprozesse in einem selbstorganisierenden System, das auf Basis dieser Entwicklungsmetapher realisiert wurde,

die selbstorganisierende Dynamik im System abbilden, kann über den Austausch bzw. der Rekonfiguration dieser Prozesse zur Laufzeit Einfluss auf die selbstorganisierende Dynamik genommen werden. Ein solcher Austausch der Koordinationsprozesse oder deren Rekonfiguration wird im Kontext dieser Arbeit als *strukturelle Adaption* bezeichnet, da dadurch die Struktur des selbstorganisierenden Verhaltens angepasst wird. In Abschnitt 4.4 wurde dabei eine Lösung präsentiert, die sich bei der strukturellen Adaption von Koordinationsprozessen an Konzepten, aber vor allem technischen Gegebenheiten des zugrundeliegenden *Coordination Space* Konstruktionsansatzes [VSL+10] orientiert. Dieser Ansatz zur strukturellen Adaption wurde im Rahmen der Entwicklung des dezentralen Koordinationsframeworks entsprechend adaptiert [PDR16c]. Abbildung 6.11 veranschaulicht nochmal den Zusammenhang zwischen selbstorganisierender Dynamik und struktureller Adaption (vgl. auch Abbildung 4.9) und den Konzepten, mit denen diese jeweils realisiert werden. Das Konzept des *Adaptation Handler*, der von den Adaptionsprozessen instruiert wird, um die lokalen Komponenten im System im Hinblick auf die selbstorganisierende Dynamik zu überwachen und ggf. den Adaptionsprozess auszulösen, wird im weiteren Verlaufs dieses Abschnitts beschrieben. Ebenso wird das Konzept des *Adaptation Topic* beschrieben, das ein Publish/Subscribe Kommunikationsmodells abbildet, über das die beteiligten Komponenten im Rahmen der Konsensfindung über eine vorgeschlagene Adaption miteinander kommunizieren.

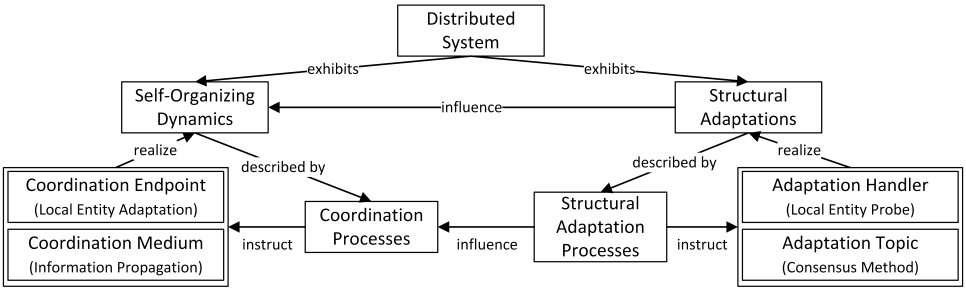



**Abbildung 6.11.:** Zusammenhang zwischen selbstorganisierender Dynamik und struktureller Adaption im Konzept des dezentralen Koordinationsrahmenwerks. Die Abbildung unterscheidet sich von Abbildung 4.9 durch die Komponenten, die zur Umsetzung der strukturellen Adaptionsprozesse instruiert werden.

Da sich die grundlegenden Konzepte des Ansatzes zur strukturellen Adaption von Koordinationsprozesse gegenüber Abschnitt 4.4 nicht weiter verändert haben, werden sie in diesem Abschnitt nicht erneut behandelt. Für einen genaueren Überblick über das Konzept der strukturellen Adaption und wie es dabei helfen kann, unerwünschten, emergenten Verhalten in selbstorganisierenden Systemen entgegenzuwirken sowie einer Beschreibung über die Einbindung des Konzepts in den Entwicklungsprozess selbstorganisierender Systeme nach [SBP+09] wird an dieser Stelle daher erneut auf Abschnitt 4.4 verwiesen. In diesem Abschnitt wird nun zunächst beschrieben, welche Eigenschaften generische Softwarekomponenten aufweisen müssen, um sie

neben der Fähigkeit zur Koordination auch mit der Fähigkeit zur strukturellen Adaption auszustatten. Im Folgenden wird dann die daraus resultierende Architekturerweiterung des dezentralen Koordinationsframeworks zur Unterstützung von strukturellen Adaptionen beschrieben. Anschließend wird auf die entwickelte Erweiterung der in Unterabschnitt 6.2.3 beschriebenen Beschreibungssprache für dezentrale Koordinationsprozesse zur Beschreibung von strukturellen Adaptionen eingegangen. Den Abschluss des Abschnittes bildet schließlich eine kurze Beschreibung des Integrations- und Entwicklungsansatz für strukturelle Adaption mit *DeCoF*.

### 6.3.1. Adaptier- und koordinierbare Komponenten

In Unterabschnitt 6.2.2 wurde auf den grundsätzlichen Unterschied zwischen *Managed*- und *Unmanaged*-Components eingegangen und erarbeitet, welcher Konzepte es bedarf, um *Unmanaged*-Components mit der Fähigkeit zur Koordination auszustatten. Ähnliche Überlegungen müssen angestellt werden, wenn es darum geht, koordinierbare, generische Softwarekomponenten mit der Fähigkeit zur strukturellen Adaption auszustatten. Das zuvor in Abschnitt 4.4 beschriebene Konzept zur strukturellen Adaption von Koordinationsprozessen verwendet spezifische *Beliefs* von BDI-Softwareagenten als lokale Performanzindikatoren, die Auskunft über die Qualität der selbstorganisierenden Dynamik liefern. Damit liefert der in Abschnitt 4.4 vorgestellte Ansatz zwar eine gute Lösung, um den Zustand von BDI-Agenten im Hinblick auf die selbstorganisierende Dynamik der Anwendung sinnvoll zu überwachen, ist in seiner Anwendbarkeit allerdings auf Multiagentensysteme, die BDI-Agenten unterstützen, limitiert. Ähnlich wie zuvor in Abschnitt 6.2 das Konzept der expliziten Koordinationsereignisse und der `ICoordinatable` Schnittstelle eingeführt wurde, werden für die strukturelle Adaption ebenfalls explizite Modellierungselemente benötigt, um den Ansatz für generische Softwarekomponenten verwendbar zu machen. Dazu wurden spezielle Adaptionselemente (*AdaptationElement*) eingeführt. Diese Elemente haben zwei Eigenschaften: Zunächst dienen sie als lokale Auslöser für den Adaptionsprozess. Dazu verfügen die Adaptionselemente über einen Eventbus (vgl. Unterabschnitt 6.2.1), über den sie potentielle *Subscriber* über Wertänderungen informieren können. Mit der Möglichkeit, Werte zu kapseln und potentiell interessierte Beobachter über Wertveränderungen zu informieren, bieten die Adaptionselemente ähnliche Eigenschaften, wie es die *Beliefs* in BDI-Agentensystemen tun und können somit in ähnlicher Weise als lokale, beobachtbare Performanzindikatoren verwendet werden. Weiterhin kapseln sie spezielle Werte, die als lokale Performanzindikatoren für das selbstorganisierende Verhalten genutzt werden können. In der aktuellen Implementierung werden dabei als Vereinfachung nur Werte im `BigDecimal` Format unterstützt, also *große* Dezimalzahlen (siehe Abbildung 6.12).

Eine weitere Erweiterung stellt das `IAdaptableCoordinatable` Interface dar, welches das bereits bekannte `ICoordinatable` Interface durch

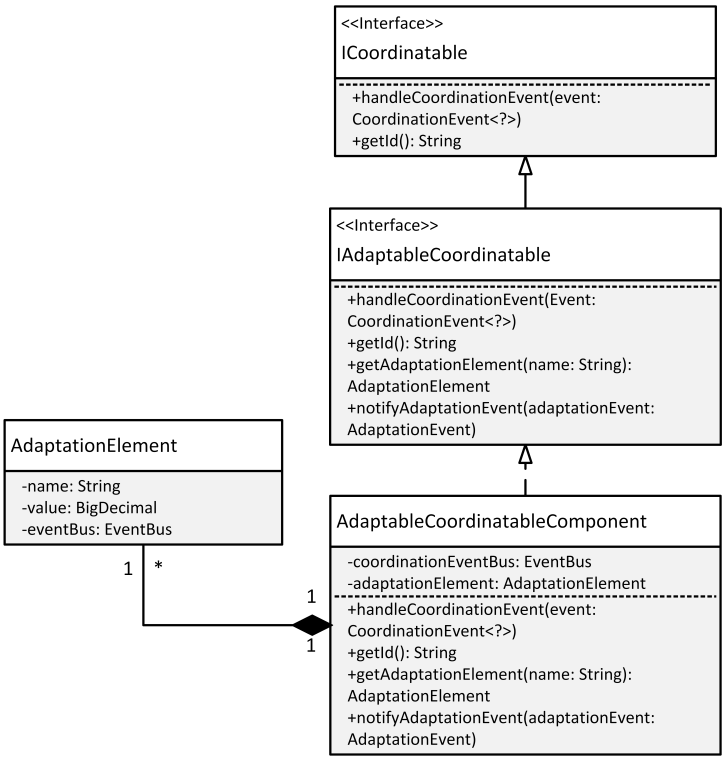



**Abbildung 6.12.:** UML Klassendiagramm zur Schnittstelle für adaptierbare und koordinierbare Komponenten.

Vererbung um zwei weitere Funktionen erweitert. Komponenten, die neben der Fähigkeit zur Koordination auch in der Lage sein sollen, an der strukturellen Adaption der Koordinationsprozesse teilzunehmen, müssen dazu die Methoden `getAdaptationElement` und `notifyAdaptationEvent` implementieren. Die erste Methode gibt für einen bestimmten Identifikator das zugehörige Adaptionselement zurück. Diese Methode wird von den sogenannten *Adaptation Handlern* benötigt, um sich bei bestimmten Adaptionselementen zu registrieren (siehe nächster Abschnitt). Die zweite Methode informiert eine Komponente über eine durchgeführte strukturelle Adaption. Diese Methode muss nur dann implementiert werden, wenn die Komponente diese Information, z.B. zur Visualisierung der durchgeführten Adaption, auch tatsächlich benötigt.

### 6.3.2. Architekturerweiterung für strukturelle Adaptionen

Um die strukturelle Adaption von Koordinationsprozessen im dezentralen Koordinationsframework zu unterstützen, wurde dessen Architektur an einigen Stellen erweitert. Strukturelle Adaptionen der Koordinationsprozesse werden dabei, genau wie die Koordinationsprozesse selber auch, in deklarativer Form beschrieben. Dabei unterscheidet sich die strukturelle Adaption von Koordinationsprozessen in *DeCoF* konzeptionell nicht von der strukturellen Adaption von Koordinationsprozessen für den *Coordination Space* Konstruktionsansatz

(vgl. Abschnitt 4.4.3). Die XML-basierte Sprache zur Beschreibung von Koordinationsprozessen, die in Abschnitt 6.2.3 vorgestellt wurde, wurde daher entsprechend erweitert, sodass in der entsprechenden Koordinationsbeschreibung auch die strukturellen Adaptionen enthalten sein können. Ein Koordinationsendpunkt, der einer koordinierbaren Komponente zugeordnet ist, liest dieses Dokument ein und erhält daraus seine Instruktionen bzgl. der Koordinationsprozesse.

Neu hinzugekommen ist, dass ein Koordinationsendpunkt für jede beschriebene, mögliche Adaption einen *Adaptation Handler* instanziiert. Ein solcher Handler kapselt die Logik, die der strukturellen Adaption zugrunde liegt und entkoppelt sie somit von der Koordinationslogik, die in den Endpunkten realisiert ist. Der Handler registriert sich dazu bei dem entsprechenden *Adaptation Element* und wird somit über Wertänderungen an dem Element benachrichtigt. Eine solche Benachrichtigung erfolgt über einem *Adaptation EventBus*, der in seiner Arbeitsweise der des *Coordination EventBus* entspricht. Wertänderungen werden dabei in Form eines *Adaptation Element Change Events* gekapselt und an die Subscriber übermittelt. Ein *Adaptation Handler* prüft nun, ob durch die Wertveränderungen die Startbedingung der strukturellen Adaption erfüllt ist. Wenn dieses der Fall ist, initialisiert der Handler einen verteilten Abstimmungsprozess, um einen Konsens darüber zu finden, ob eine Mehrheit der Komponenten der Adaption zustimmt. Abbildung 6.13 stellt die ursprüngliche Architektur des dezentralen Koordinationsframeworks und der Erweiterung für strukturelle Adaptionen dar.

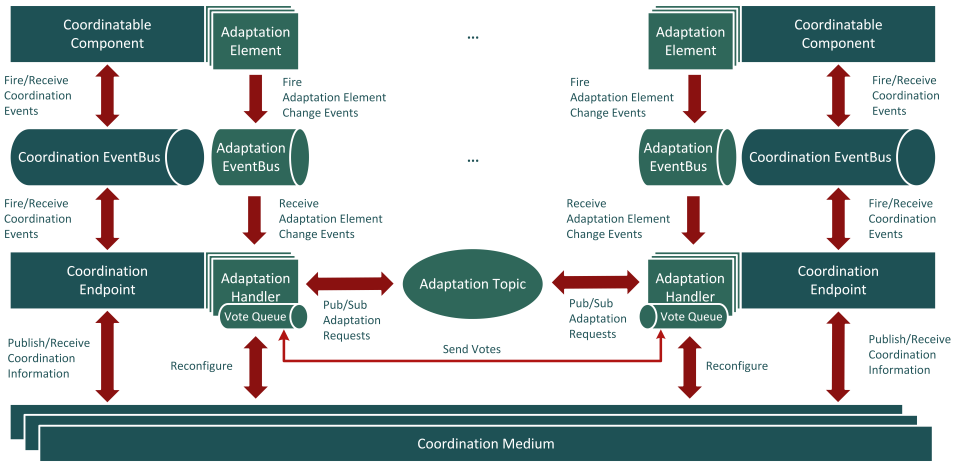



**Abbildung 6.13.:** Architektur des dezentralen Koordinationsframeworks mit Unterstützung für strukturelle Adaptionen - nach (PDR16c). Die Elemente in der Mitte, die im helleren Farbton abgebildet sind, stellen die Erweiterungen zur Unterstützung von strukturellen Adaptionen dar. Die Elemente am Rande, die im dunkleren Farbton abgebildet sind, zeigen die Elemente die bereits vorhandene Architektur des Frameworks (vgl. Abbildung 6.5).

Wie bereits in Abschnitt 4.4 beschrieben, basieren die strukturellen Adaptionen auf einem antizipierenden Ansatz, bei dem mögliche Adaptionen bereits zur Entwurfszeit von einem Entwickler beschrieben werden, um dann zur Laufzeit automatisiert ausgeführt zu werden, wenn die entsprechenden

Bedingungen eintreten, die eine solche Adaption nötig machen. Die Bedingungen referenzieren dabei lokale Zustände der Komponenten. Da sich die Zustände der Komponenten dabei unterscheiden können, also einige Komponenten bzgl. der selbstorganisierenden Dynamik einem ungenügenden Verhalten ausgesetzt sein können, während andere es nicht sind, muss ein Konsens darüber gefunden werden, ob die Adaption durchgeführt werden soll oder nicht. Dazu wird der verteilte Abstimmungsprozess verwendet, dessen technische Realisierung sich wie folgt gestaltet: Es wird für jede mögliche Adaption ein verteiltes *Adaptation Topic* erstellt. Zur Realisierung des Publish/Subscribe Kommunikationsmodells wird dabei *Hazelcast* [Joh13] als Infrastruktur verwendet. Mit *Hazelcast* lassen sich auf dem Peer-to-Peer Modell basierende Kommunikationsinfrastrukturen aufbauen. Die *Adaptation Handler* subskribieren nun alle das entsprechende *Adaptation Topic* und publizieren hier Abstimmungsaufforderungen, um eine strukturelle Adaption vorzuschlagen. Wenn ein *Adaptation Handler* eine solche Aufforderung über das verteilte Topic erhält, evaluiert er sie gegen den Wert seines lokalen *Adaptation Elements* und entscheidet auf Basis dieses lokalen Zustands darüber, ob er der vorgeschlagenen Adaption zustimmt oder nicht. Dabei wird das bereits in Abschnitt 4.4.3 beschriebene Schwellwert-Konzept verwendet, mit dem Bedingungen formuliert werden, die nicht so streng sind wie die Bedingungen, die dafür sorgen würde, dass ein *Adaptation Handler* seinerseits den Adaptionsprozess auslöst, die jedoch einen negativen Trend abbilden, der einen Handler dazu veranlasst, einer vorgeschlagenen Adaption zuzustimmen. Jeder *Adaptation Handler* ist mit einer Abstimmungswarteschlange ausgestattet, die von den anderen Handlern verwendet wird, um ihre Abstimmungsergebnisse (*Vote Queue*) an den Handler zurück zu schicken, der den Abstimmungsprozess initiiert hat. Der Initiator des Abstimmungsprozesses startet die Auswertung der Abstimmungsergebnisse, sobald er die benötigte Anzahl an Antworten erhalten hat oder ein Timeout aufgetreten ist. Wenn die notwendige Mehrheit erreicht wurde, publiziert er die entsprechende Adaptionsanweisung über das *Adaptation Topic* an alle anderen *Adaptation Handler*. Gemäß dieser Anweisung passen die *Adaptation Handler* dann ihre lokalen Instanzen der Koordinationsmedien an.

Resultierend aus dieser Schichtenarchitektur zeigt Abbildung 6.14 in Form eines UML Klassendiagramms, aus welchen Klassen sich das Koordinationsrahmenwerk inklusive der Erweiterung um die Fähigkeit zur strukturellen Adaption zusammensetzt und wie die Klassen dabei miteinander interagieren.

### 6.3.3. Beschreibung von Adaptionen

Im Folgenden wird beschrieben, wie die in Abschnitt 6.2.3 vorgestellte Sprache zur deklarativen Beschreibung von Koordinationsprozessen um deklarative Beschreibungsmöglichkeiten für deren strukturelle Adaption erweitert wurde. Konzeptionell orientiert sich diese Spracherweiterung dabei an der in Ab-

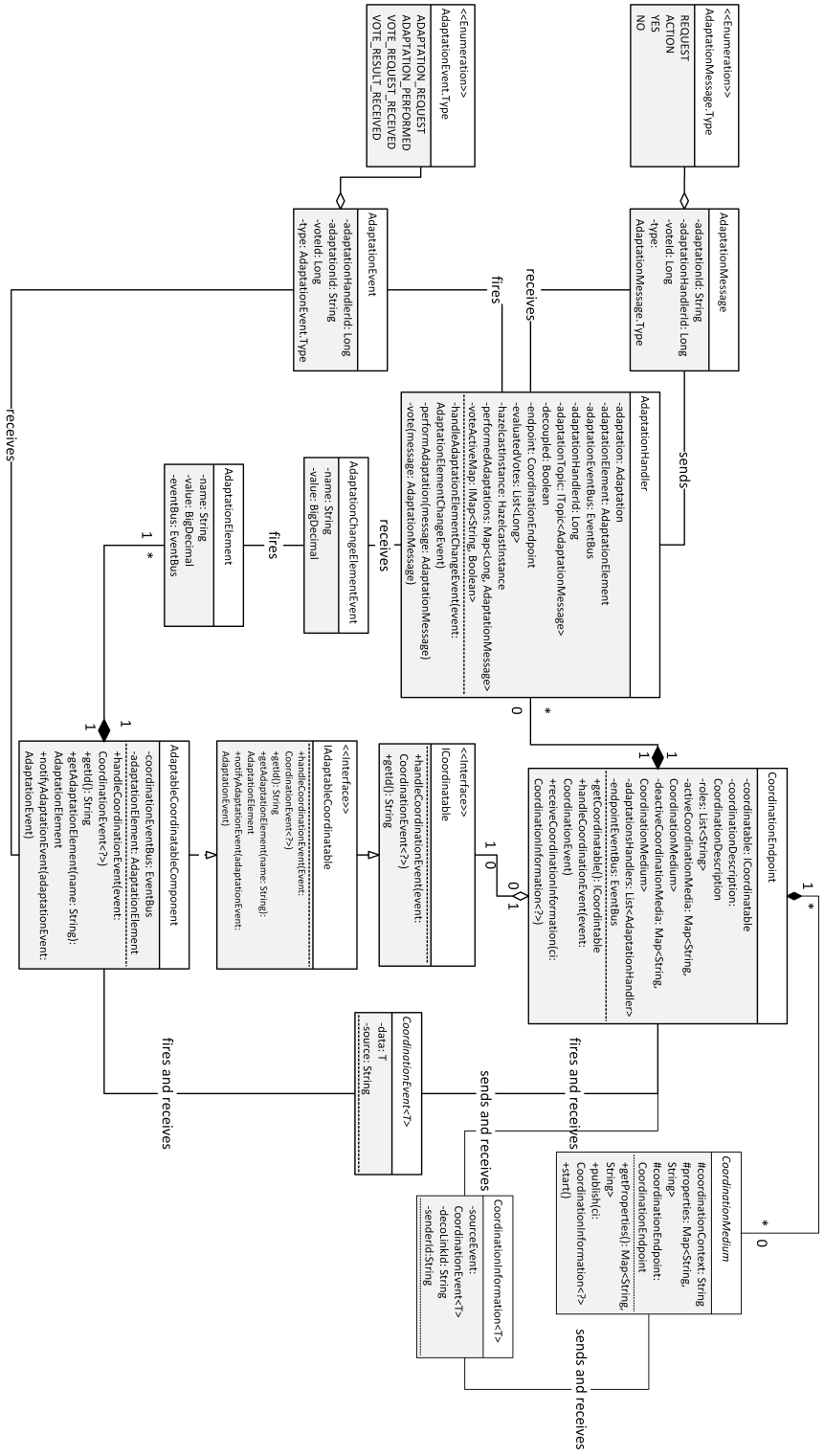



**Abbildung 6.14.:** UML Klassendiagramm des dezentralen Koordinationsframeworks mit den wichtigsten Klassen für die strukturellen Adaption.

schnitt 4.4.3 beschriebenen Spracherweiterung der *MASDynamics* Sprache zur Beschreibung von strukturellen Adaptionen für den *Coordination Space* Konstruktionsansatz.

Abbildung 6.15 stellt diese Erweiterung der *DeCoF* Koordinationsbeschreibungssprache in Form eines UML Klassendiagramms dar. Koordinationsbeschreibungen (`CoordinationDescription`) können dabei durch eine Reihe von möglichen Adaptionen (`Adaptation`) erweitert werden. Die Beschreibung einer strukturellen Adaption beinhaltet dabei zunächst einen Identifikator in Form einer Zeichenkette (`id`). Außerdem wird hier über den Namen die Rolle der Komponenten referenziert (`role`), die an dieser Adaption beteiligt ist. Weiterhin muss der Name des Adaptionselements angegeben werden (`element`), das als lokaler Performanzindikator für die Adaption verwendet werden soll. Ferner kann über Boolean-Flags spezifiziert werden, ob die Adaption nur ein einziges Mal oder ggf. mehrmals durchgeführt werden soll (`single`) und ob der Adaptionsvorgang entkoppelt vom Ausführungsthread der Komponente in einem eigenen Thread durchgeführt werden soll (`decoupled`). Die Entkopplung des Adaptionsvorgangs vom Ausführungsthread der Komponente entspricht dabei konzeptionell der Möglichkeit der Entkopplung des Koordinationsvorgangs vom Ausführungsthread, wie er in Abschnitt 6.2.3 beschrieben wurde. Schließlich beinhaltet die Beschreibung einer Adaption noch die drei Elemente `Trigger`, `Voting` und `Action`.

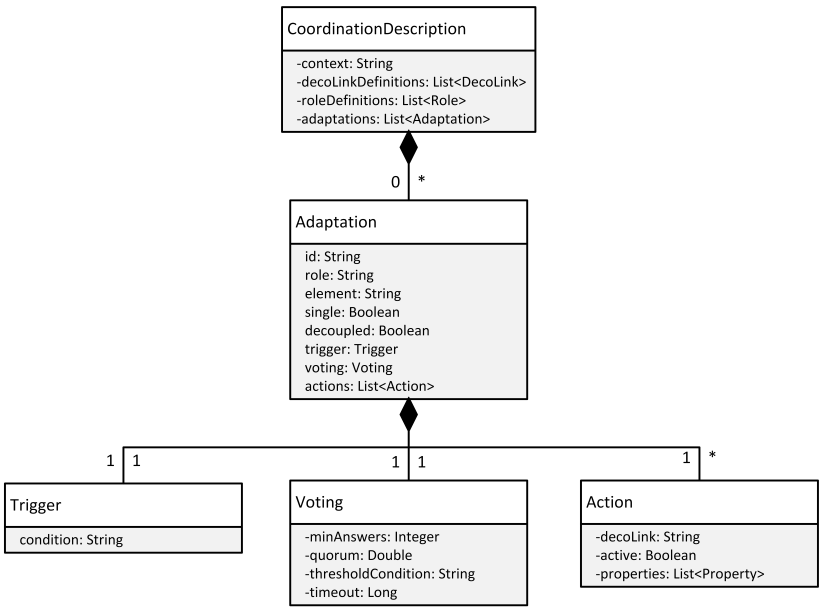



**Abbildung 6.15.:** UML Klassendiagramm der Erweiterung der DeCoF Koordinationsbeschreibungssprache für strukturelle Adaptionen.

In der `Trigger` Klasse ist dabei der Auslöser des Adaptionsvorgangs beschrieben. Hier wird eine Bedingung (`condition`) spezifiziert, die das

entsprechende Adaptionselement referenziert. Die Bedingung kann dabei einfache mathematische oder Bool'sche Ausdrücke beinhalten[5].

Innerhalb der `Voting` Klasse wird das Abstimmungsverfahren konfiguriert. Dabei kann zunächst die minimale Anzahl an Antworten (`minAnswers`) angegeben werden, die der Initiator des Abstimmungsprozesses erwartet, ehe er mit der Auswertung der Ergebnisse beginnt. Außerdem kann ein Timeout in Millisekunden angegeben werden (`timeout`), nach dessen Ablauf der Initiator in jedem Fall mit der Auswertung der Abstimmungsergebnisse beginnt. Neben diesen beiden optionalen Parametern muss in jedem Fall angegeben werden, wie viel Prozent der eingehenden Antworten (`quorum`) positiv ausfallen müssen, damit eine vorgeschlagene Adaption durchgeführt wird. Mit der `thresholdCondition` kann die bereits beschriebene Schwellwert-Bedingung ausgedrückt werden, die einen negativen Trend im Bezug auf die selbstorganisierende Dynamik abbildet. Wenn diese Bedingung eintritt, stimmt eine Komponente einer vorgeschlagenen Adaption zu.

In der `Action` Klasse schließlich wird beschrieben, in welcher Form die vorgeschlagene Adaption durchgeführt wird. Dabei kann zwischen zwei Arten der Anpassung unterschieden werden: Der parametrischen und der strukturellen. Bei einer parametrischen Adaption werden nur Parameter des Koordinationsprozesses angepasst, während bei einer strukturellen Adaption die Struktur der Koordinationsprozesse an sich verändert wird. In jedem Fall wird über das Feld `decoLink` der Identifikator des dezentralen Koordinationsprozesses, der angepasst werden soll, referenziert. Handelt es sich um eine parametrische Adaption, können über das `properties` Feld die Parameter des entsprechenden Koordinationsprozesses verändert werden. Dabei wird die gleiche *Key-Value*-Paar Referenzierung verwendet, wie sie in der Koordinationsmedium Klasse verwendet wird (vgl. Abschnitt 6.2.3). Wenn es sich bei der Adaption um eine strukturelle handelt, kann über das `active` Feld angegeben werden, ob der entsprechende Koordinationsprozess aktiviert bzw. deaktiviert werden soll.

Der `Adaptation`-Klasse zur Beschreibung der Adaptionen ist dabei in jedem Fall mindestens eine Aktion (`Action`) zugeordnet. Eine Adaption kann aber auch eine Reihe von Aktionen zur Folge haben, die unterschiedliche Koordinationsprozesse in unterschiedlicher Art und Weise beeinflussen. Hierbei ist auch eine eine Mischung aus parametrischer und struktureller Adaption möglich, bei der bei einigen Koordinationsprozessen nur die Parameter angepasst werden, während andere Prozesse komplett aktiviert bzw. deaktiviert werden.

### 6.3.4. Integrations- und Entwicklungsansatz

Grundsätzlich folgt der Integrationsansatz der strukturellen Adaption im Rahmen von *DeCoF* dem Entwicklungsansatz der strukturellen Adaption

[5] Implementierungsdetail: Zur Auswertung der Ausdrücke wird der *EvalEx - Java Expression Evaluator* verwendet (`https://github.com/uklimaschewski/EvalEx`, aufgerufen am 1. Februar 2017).

von Koordinationsprozessen, wie er bereits in Abschnitt 4.4.1 beschrieben wurde. Somit knüpft er eng an den Entwicklungsansatz dem *SodekoVS* Projekt [SBP+09] an. Dieser Ansatz fordert, dass in der ersten Phase, der Anforderungsphase, das gewünschte Systemverhalten zu spezifizieren ist. In der folgenden Analysephase werden dazu passenden Koordinationsmetaphern identifiziert und kombiniert, sodass das gewünschte, adaptive Verhalten erzielt werden kann. In der Design-Phase werden aus diesen Metaphern die entsprechenden Koordinationsstrategien für die Realisierung abgeleitet. In der Implementierungsphase werden diese umgesetzt und schließlich in der Test-Phase das Systemverhalten durch Simulation validiert. Wenn während dieser Phase oder auch als Ergebnis der initialen Analyse beobachtet wird, dass bestimmte Koordinationsprozesse oder ihre Konfigurationen unter bestimmten Bedingungen zwar das vorgesehene Systemverhalten zeigen, unter anderen Bedingungen aber zu einem ineffizienten Verhalten führen, dann sind dieses mögliche Bedingungen für den Ansatz einer strukturellen Adaption der Koordinationsprozesse. Die größte Herausforderung aus Entwicklersicht ist hierbei, genau die Bedingungen zu identifizieren und entsprechend zu modellieren, die eine strukturelle Adaptation erfordern und daraus folgend ihre Abbildung auf die Performanz-Indikatoren sowie schließlich die Modellierung einer geeigneten strukturellen Adaption.

In Anlehnung an die in Abschnitt 6.2.2 beschriebenen vier Schritte, um auf Basis von *DeCoF* eine Anwendung mit dezentralen Koordinationsprozessen auszustatten, ergeben sich im Fall einer strukturellen Adaption folgende drei weitere Schritte:

5. Identifikation der Bedingungen oder Zustände im System, die dazu führen, dass das System nicht mehr das vorgesehene selbstorganisierende Verhalten zeigt.

6. Identifikation der Koordinationsprozesse bzw. deren Komposition, die unter diesen Bedingungen nicht mehr zu dem gewünschten vorgesehenen selbstorganisierenden Verhalten führen und entsprechender Maßnahmen, wie diesem durch eine Veränderung der Koordinationsprozesskomposition oder eine Veränderung an den Koordinationsprozessparametern entgegengewirkt werden kann.

7. Formulierung entsprechender Adaptationsanweisungen (vgl. Abschnitt 6.3.3), welche die identifizierten Bedingungen oder Zustände als Adaptionsbedingungen enthalten und Bezug nehmen auf die Koordinationsprozesse und die entsprechenden Anpassungen an diesen.

Die Modellierung einer strukturellen Adaption erfordert also keinen weiteren programmiertechnischen Aufwand. Vielmehr besteht die Herausforderung dabei, geeignete Bedingungen und Adaptionen zu finden und diese entsprechend in deklarativer Art und Weise zu formulieren. Daher bietet sich in Anlehnung an das *SodekoVS* Projekt [SBP+09] ein iteratives, simulationsgetriebenes Vorgehensmodell an, bei dem auf Basis von Simulationen diese

Bedingungen und Adaptionen explorativ bestimmt und verfeinert werden. Das entwickelte Rahmenwerk unterstützt einen Entwickler also soweit, als dass es ein fertiges Gerüst für die strukturellen Adaptionen zur Verfügung stellt, dass nur noch entsprechend in seiner Ausführung instruiert werden muss.

## 6.4. Zusammenfassung

In diesem Kapitel ging es darum, einen neuartigen Ansatz zur Realisierung von dezentraler Koordination in verteilten Systemen vorzustellen. Für die Realisierung selbstorganisierender Eigenschaften in einem verteilten System ist die Fähigkeit zur Koordination ein integraler Bestandteil. Erst durch die sinnvolle Koordination der beteiligten Komponenten ist es möglich, ein kohärentes, adaptives Verhalten zu erzielen. Im vorangegangen Kapitel wurde ein Überblick über skalierbare, verteilte und adaptive Co-Simulationssysteme gegeben. Dabei wurde u.a. deutlich, dass eine essentielle Herausforderung bei der Realisierung solcher Systeme die Heterogenität der verwendeten Technologien und Plattformen ist. Dabei wurde ebenfalls deutlich, dass diese Heterogenität zwar für neue Anforderungen im Bereich der Interoperabilität sorgt, es aber gleichzeitig auch gute Gründe dafür gibt, in diesem Umfeld auf unterschiedliche Technologien und Ansätze zur Modellbildung der Simulation zu setzen. Aufgrund dieser Heterogenität können Koordinationsansätze wie *DeCoMAS* [SR09a] oder *Coordination Space* [VSL+10], die im vorherigen Verlauf dieser Arbeit präsentiert wurden, aufgrund ihrer Fokussierung auf die Koordination von Multiagentensystemen nicht verwendet werden. Daher wurden in diesem Kapitel Ansätze präsentiert, die einige Konzepte aus diesen thematischen Vorarbeiten aufgreifen und sie auf ein allgemeineres Konzept, dass die Konstruktion selbstorganisierender Systeme unabhängig von der verwendeten Systemarchitektur erlaubt, adaptieren. Dazu wurde zunächst ein Konzept beschrieben, das die dezentrale Koordination in verteilten Systemen, die auf Basis des Aktiven Komponenten Konstruktionsansatzes realisiert sind, unterstützt. Dieses Konzept diente dabei als Zwischenschritt, der den Übergang von der Koordination von Multiagentensystemen im Speziellen zu verteilten Systemen im Allgemeinen vorbereitet.

Darauf aufbauend wurde in Abschnitt 6.2 ein dezentrales Koordinationsframework präsentiert, das es erlaubt, heterogene Systeme unabhängig von der zugrundeliegenden Architektur zu koordinieren. Dabei wurden primär zwei Forschungsfragen beantwortet:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen um koordinierbar zu sein?
2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

Die erste Forschungsfrage wurde dabei zunächst konzeptuell in Unterabschnitt 6.2.1 mit der Einführung expliziter Koordinationsereignisse beant-

wortet. Wenn es aufgrund einer heterogene Systemarchitekturen keine einheitlichen, impliziten Ereignisse bzw. Zustände gibt, die eine Koordinationsschicht im Hinblick auf für die Koordination relevante Ereignisse oder Zustände überwachen kann, dann müssen solche expliziten Ereignisse verwendet werden. In Unterabschnitt 6.2.1 wurde daher zunächst auf architektonischer Ebene beschrieben, wie eine möglichst lose Kopplung zwischen den koordinierbaren Komponenten, die diese expliziten Koordinationsereignisse auslösen, und der Koordinationsschicht, die auf diese Ereignisse reagiert, realisiert werden kann. Durch die Verwendung eines sogenannten *Eventbus* konnte an dieser Stelle eine möglichst lose Kopplung erzielt werden, bei der die Komponenten zwar Koordinationsereignisse auslösen müssen, sie allerdings keinerlei Kenntnis über die An- bzw. Abwesenheit einer observierenden Koordinationsschicht benötigen. Dabei wurde das bereits bekannte Konzept der Koordinationsendpunkte aus *DeCoMAS* und dem *Coordination Space* Konzept übernommen. Diese Endpunkte kapseln alle Aktivitäten, die für die Koordination relevant sind und agieren unabhängig von den Komponenten, die die eigentliche Anwendungslogik implementieren. In Unterabschnitt 6.2.2 wurde dann die erste Forschungsfrage auf technischer Ebene beantwortet, indem der Integrationsansatz des erarbeiteten dezentralen Koordinationsframeworks präsentiert wurde. Dabei wurde mit dem `ICoordinatable` Interface die Schnittstelle präsentiert, die Komponenten implementieren müssen, wenn sie mit der Fähigkeit zur Koordination ausgestattet werden sollen. Außerdem wurde auf technischer Ebene die Verwendung bzw. Handhabung von Koordinationsereignissen durch die Komponenten beschrieben.

Die zweite Forschungsfrage wurde ebenfalls in Unterabschnitt 6.2.1 in Form der präsentierten Architektur beantwortet. Dabei wurde auf konzeptioneller Ebene gezeigt, wie die Koordination von generischen Softwarekomponenten auf Basis der Konzepte Koordinationsereignisse, Koordinationsbus, Koordinationsendpunkte und Koordinationsmedien erfolgen kann. Die technische Ebene der Forschungsfrage wurde dann wiederum in Unterabschnitt 6.2.2 in Form des Integrationsansatzes des präsentierten Rahmenwerks beantwortet. Abgerundet wurde die Beantwortung der zweiten Forschungsfrage durch Unterabschnitt 6.2.3, in dem beschrieben wurde, wie konkret Koordinationsprozesse mit dem Rahmenwerk realisiert und beschrieben werden können. Dabei wurde sowohl der konzeptuelle Ansatz der Koordinationsprozesse im Kontext des Rahmenwerks als auch die deklarative Sprache zur Beschreibung von Koordinationsprozessen vorgestellt.

Im letzten Abschnitt dieses Kapitels ging es schließlich darum, wie das Konzept der strukturellen Adaption von Koordinationsprozessen, das in Abschnitt 4.4 im Kontext des *Coordination Space* Konstruktionsansatzes erarbeitet wurde, adaptiert werden kann, damit es im Kontext des dezentralen Koordinationsframeworks für generische Softwarekomponenten angewendet werden kann. Dabei wurde gezeigt, dass der Ansatz grundsätzlich ohne Veränderungen an den grundlegenden Konzepten adaptiert werden kann. Ähnlich wie bei der Umsetzung der Koordinationsfunktionalität bedarf es

aber auch hier der Verwendung expliziter Modellierungselemente, um den Ansatz für generische Softwarekomponenten nutzbar zu machen. Daher wurde in Abschnitt 6.3 primär das Konzept der Adaptionselemente vorgestellt, ehe die Erweiterung des dezentralen Koordinationsframeworks zur Unterstützung von strukturellen Adaptionen auf architektonischer und technischer Ebene beschrieben wurde.

Abschließend lässt sich feststellen, dass in diesem Kapitel aufgezeigt wurde, wie die Koordinationskonzepte, die in den vorangegangenen Teilen dieser Arbeit für Multiagentensysteme präsentiert wurden, generalisiert werden können, sodass es möglich ist, auf Basis dieser Konzepte die Koordination sowie die strukturelle Adaption der Koordinationsprozesse für generische Softwarekomponenten nutzbar zu machen.

# 7. Anwendung und Bewertung

In diesem Kapitel wird zunächst anhand zweier Anwendungsstudien beschrieben, wie dezentrale Koordinationsprozesse dazu verwendet werden können, um selbstorganisierendes Verhalten umzusetzen. Dabei geht es insbesondere, darum das im Rahmen dieser Dissertation entwickelte dezentrale Koordinationsframework sowie das Konzept der strukturellen Adaption von Koordinationsprozessen zu validieren. Unter dem Begriff der Validierung versteht man im Bereich der Softwarequalitätssicherung die Prüfung der Eignung einer Software bezogen auf ihren Einsatzzweck [Bal11]. Dementsprechend wird im ersten Abschnitt dieses Kapitels zunächst das Konzept der strukturellen Adaption von Koordinationsprozessen auf seine Eignung zur Überwindung von dynamischen Ineffizienzen in einem selbstorganisierenden System überprüft. Anschließend wird das entwickelte dezentrale Koordinationsframework auf seine Eignung im Bezug auf die Realisierung von dezentralen Koordinationsprozessen untersucht werden. Im zweiten Abschnitt dieses Kapitels erfolgt dann eine Bewertung der in dieser Dissertation erzielten Ergebnisse. Dazu werden zunächst existierende Ansätze mit einer ähnlichen Zielsetzung vorgestellt, um diese dann mit dem entwickelten Koordinationsframework zu vergleichen. Schließlich erfolgt eine abschließende Bewertung, indem die in der Einleitung formulierten Forschungsfragen beantwortet werden.

## 7.1. Anwendungen

In diesem Abschnitt werden zwei Anwendungsstudien betrachtet, die im Rahmen dieser Dissertation durchgeführt wurden. Dabei wird zunächst die *Mars-World Mining Operation* Anwendungsstudie [PRD15] präsentiert. Hier wird das Konzept der strukturellen Adaption von Koordinationsprozessen, wie es in Abschnitt 4.4 vorgestellt wurde, angewandt, um die Koordinationsprozesse zwischen autonomen Robotern, die auf dem Mars Erz abbauen, automatisch zu adaptieren. Dabei kommt es zu einer strukturellen Adaption der Koordinationsprozesse, sobald die vorgesehene, selbstorganisierende Dynamik aufgrund von externen Umwelteinflüssen nicht mehr erzielt werden kann. In der zweiten Anwendungsstudie [PDR16b, PDR16c] wird das selbstorganisierende Fahrradverleihsystem aus Abschnitt 2.5.2 wieder aufgegriffen, um die Verwendung des dezentralen Koordinationsframeworks, das in Abschnitt 6.2 präsentiert wurde, aufzuzeigen. Dabei wird sowohl darauf eingegangen, wie sich dezentrale Koordinationsprozesse mit Hilfe der entwickelten deklarativen Beschreibungssprache modellieren lassen, als auch darauf welche technischen Integrationsschritte von einem Entwickler durchgeführt werden müssen, um

generische Softwarekomponenten mit der entsprechenden Fähigkeit zur Koordination auszustatten. Weiterhin wird dabei gezeigt, wie das Konzept der strukturellen Adaption von Koordinationsprozessen mit dem dezentralen Koordinationsframework umgesetzt werden kann.

### 7.1.1. MarsWorld Mining Operation

Das *MarsWorld* Szenario ist ein hypothetisches Anwendungsszenario, das zuerst in [Fer99] präsentiert und in [PRD15] im Rahmen dieser Dissertation aufgegriffen wurde, um das Konzept der strukturellen Adaptionen von Koordinationsprozessen (vgl. Abschnitt 4.4) zu veranschaulichen. Dabei wird eine Gruppe von autonomen Robotern zum Planeten Mars geschickt, um dort Erz abzubauen. Der Abbauprozess besteht dabei aus drei Aktivitäten:

1. *Analyse* von potentiellen Erzvorkommen, um das Vorhandensein von Erz zu verifizieren.

2. Abbau oder *Produktion* der analysierten Erzvorkommen.

3. *Transport* des abgebauten Erzes zu einer Heimatbasis.

Die Roboter werden in diesem Szenario alle durch Software-Agenten kontrolliert und sind darauf spezialisiert, eine der drei Abbau-Operationen auszuführen. Sogenannte *Sentry* Agenten sind mit ausgeklügelten Sensoren ausgestattet, um potentielle Erzvorkommen zu analysieren. *Producer* Agenten besitzen die Fähigkeit, Erz an analysierten Vorkommen abzubauen und *Carry* Agenten können das abgebaute Erz dann zurück zu der Heimatbasis transportieren. Da keiner der drei Agententypen alleine dazu in der Lage ist, den Abbauprozess durchzuführen, müssen sie zusammenarbeiten, um das kollaboratives Ziel zu erreichen. Daher wurde dieses Szenario als Anwendungsstudie ausgewählt, da es erforderlich ist, die Agenten zu koordinieren, um das gemeinsame Ziel zu erreichen. Dabei werden die *Sentry* Agenten die potentiellen Erzvorkommen analysieren, um dann die *Producer* Agenten zu informieren, sollten sie Erz gefunden haben. Die *Producer* bauen daraufhin das Erz ab und informieren die *Carry* Agenten, damit diese das abgebaute Erz zurück zur Heimatbasis transportieren. Initial explorieren alle Agenten zufällig die Umgebung. Dazu sind alle Agenten mit Sensoren ausgestattet, sodass sie in der Lage sind, potentielle Erzvorkommen zu entdecken. Wenn *Producer* oder *Carry* Agenten auf potentielle Erzvorkommen stoßen, informieren sie die *Sentry* Agenten darüber. *Sentry* Agenten, die auf eine potentielle Erzquelle gestoßen sind oder von anderen Agenten über ein potentielles Vorkommen informiert wurden, bewegen sich dorthin und analysieren dieses. Wenn das Vorhandensein von Erz verifiziert werden konnte, fragen sie einen *Producer* Agenten an, damit dieser das Vorkommen abbaut. Dementsprechend fragen die *Producer* einen *Carry* Agenten an, nachdem sie das Erz abgebaut haben, damit dieser es zurück zur Heimatbasis transportieren.

### Koordination

Da das *MarsWorld* Szenario keine vordefinierten organisatorischen Strukturen aufweist, müssen die Agenten in irgendeiner Form koordiniert werden, damit sie in der Lage sind, ihr kollaboratives Ziel zu erreichen. Aus der eingangs erwähnten Kommunikation zwischen den drei Agententypen lassen sich drei Koordinationsprozesse ableiten, welche die Verteilung der benötigten Informationen zwischen den Agenten realisieren:

1. Koordinationsinformationen, die *Producer* und *Carry* Agenten zu den *Sentry* Agenten schicken, sobald sie auf potentielle Erzvorkommen stoßen, während sie die Umgebung explorieren.

2. Koordinationsinformationen, die die *Sentry* Agenten zu den *Producer* Agenten schicken, sobald sie ein Erzvorkommen analysiert und dort tatsächlich Erz gefunden haben.

3. Koordinationsinformationen, welche die *Producer* Agenten zu den *Carry* Agenten schicken, um diese über abgebautes Erz, das zur Heimatbasis transportiert werden kann, zu informieren.

Damit der Einfluss von sowohl den selbstorganisierenden Koordinationsprozessen als auch ihrer strukturellen Adaptionen untersucht werden kann, wurden zwei unterschiedliche Ausprägungen für die drei beschriebenen Koordinationsprozesse realisiert. Die erste Ausprägung basiert dabei auf einer einfachen, zufälligen Auswahl und weist daher kein selbstorganisierendes Verhalten auf. In diesem Fall wählt jeder der drei Koordinationsprozesse zufällig einen Agenten des entsprechenden Typs aus und informiert diesen über das gefundene, analysierte oder abgebaute Erz.

Die zweite Ausprägung basiert auf einem *Nachbarschafts*-Ansatz, bei dem jeder Koordinationsprozess den entsprechenden Agenten (benötigter Typ) auswählt, der nächstgelegen zum sendenden Agenten ist. Dadurch, dass immer der nächstgelegene Agent ausgewählt wird, wird sichergestellt, dass die Agenten minimale Distanzen zurücklegen müssen, um ihre designierten Ziele zu erreichen. Wenn die Umgebung aus mehreren Gruppen von Erzvorkommen besteht, die sich dadurch auszeichnen, dass eine geringe Distanz zwischen den Vorkommen innerhalb der Gruppe besteht und eine große Distanz zu den anderen Gruppen, dann führt dieser Koordinationsansatz dazu, dass die Agenten auf emergente Weise selbstorganisierende lokale Abbau-Gruppen bilden. Offensichtlich wird für diese Art der Koordination Wissen über die Positionen der einzelnen Agenten benötigt. Daher wird die Umgebung in diesem Beispiel mit einem Positions-Service ausgestattet, der diese Informationen bereitstellt. Die drei Koordinationsprozessausprägungen, die auf diesem Ansatz basieren, greifen dabei auf diesen Service zu.

Aus den beschriebenen Koordinationsansätzen ergeben sich folgende drei Koordinationsprozesse für die zufällige Auswahl:

**latest_target_seen_random:** Der Koordinationsprozess informiert einen *Sentry* Agenten, sobald ein *Producer* oder *Carry* Agent auf ein potentielles Erzvorkommen gestoßen sind. Dabei wird zufällig einer der *Sentry* Agenten in der Umgebung selektiert. Dieser fügt die empfangene Position seiner Liste der zu analysierenden Vorkommen hinzu.

**latest_target_analyzed_random:** Der Koordinationsprozess wählt zufällig einen *Producer* Agenten aus, sobald ein *Sentry* Agent ein potentielles Erzvorkommen analysiert und dort Erz gefunden hat. Der *Producer* Agent fügt die empfangene Position seiner Liste der abzutragenden Vorkommen hinzu.

**latest_target_produced_random:** Dieser Koordinationsprozess wählt zufällig einen *Carry* Agenten aus, wenn ein *Producer* Agent Erz an einem Vorkommen abgearbeitet hat und informiert diesen über die entsprechende Position. Der *Carry* Agent fügt die Position seiner Liste der abzutransportieren Erzpositionen hinzu.

Entsprechend bilden folgende drei Koordinationsprozesse den Nachbarschafts-Ansatz ab:

**latest_target_seen_nearest:** Immer, wenn ein *Producer* oder *Carry* Agent auf ein potentielles Erzvorkommen stoßen, wählt dieser Koordinationsprozess den nächstgelegenen *Sentry* Agenten aus und informiert diesen über die Position des potentiellen Vorkommens. Der *Sentry* Agent fügt die empfangene Position seiner Liste der zu analysierenden Vorkommen hinzu.

**latest_target_analyzed_nearest:** Nachdem ein *Sentry* Agent ein potentielles Erzvorkommen analysiert hat und dort tatsächlich Erz gefunden hat, informiert dieser Prozess den nächstgelegenen *Producer* Agenten. Der *Producer* Agent fügt die empfangene Position seiner Liste der abzutragenden Vorkommen hinzu.

**latest_target_produced_nearest:** Wenn ein *Producer* Agent ein Erzvorkommen abgetragen hat, wählt dieser Koordinationsprozess den nächstgelegenen *Sentry* Agent aus und informiert diesen, sodass der *Sentry* Agent das abgebaute Erz zurück zur Heimatbasis transportieren kann.

Quelltext 7.1 zeigt beispielhaft die deklarative Beschreibung des `latest_target_seen_nearest` Koordinationsprozesses. Die Beschreibung erfolgte in der bereits beschriebenen *MASDynamics* Sprache [SR09b]. Der Quelltext zeigt, dass immer, wenn ein `callSentryEvent` in einem *Producer* oder *Carry* Agenten observiert wird, entsprechende Koordinationsinformationen über das zuletzt gefundene Ziel zu einem *Sentry* Agenten kommuniziert werden. Der entsprechende Event wird dabei innerhalb eines *Producer* oder *Carry* Agenten ausgelöst, wenn eine potentielle Erzquelle gefunden wurde. Basierend auf dem spezifizierten Koordinationsmechanismus (`sodekovs.marsworld.coordination.NearestMechanism`) wird

der nächstliegende *Sentry* Agent als Empfänger selektiert. Der Koordinationsmechanismus ist als Java-Klasse implementiert und verwendet den Positions-Service, der von der Umgebung angeboten wird, um die Position des *Sentry*-Agenten in Erfahrung zu bringen. Wenn der *Sentry* Agent die Koordinationsinformationen mit der Position der potentiellen Erzquelle empfängt, wird der spezifizierte `latestTargetEvent` vom *Coordination Enactment* des *Sentry* Agenten ausgelöst. Das hat zur Folge, dass die Position des Erzvorkommens in die Liste der zu analysierenden Positionen hinzugefügt wird. Die anderen Koordinationsprozessbeschreibungen sind entsprechend aufgebaut.

```
<realization id="latest_target_seen_nearest">
 <from>
  <agent_element element="callSentryEvent" agent_id="Producer" type="
      INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="latest_target"/>
   </parameter_mappings>
  </agent_element>
  <agent_element element="callSentryEvent" agent_id="Carry" type="
      INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="latest_target"/>
   </parameter_mappings>
  </agent_element>
 </from>
 <mechanism_configuration
  mechanism_id="sodekovs.marsworld.coordination.NearestMechanism" agent_type
      ="JADEX">
 </mechanism_configuration>
 <to>
   <agent_element element="latestTargetEvent" agent_id="Sentry" type="
       INTERNAL_EVENT">
   <parameter_mappings>
    <mapping ref="latest_target" name="latest_target"/>
   </parameter_mappings>
  </agent_element>
 </to>
 <active>true</active>
</realization>
```

**Quelltext 7.1:** Deklarative Beschreibung der *latest_target_seen_nearest* Koordinationsprozess-Realisierung in der *MASDynamics* Sprache.

### Strukturelle Adaption

Betrachtet man die soeben vorgestellten Koordinationsprozess-Ausprägungen, so wird schnell klar, dass der Nachbarschafts-Ansatz besser geeignet ist, da er die Formation von lokalen Abbau-Gruppen unterstützt. Dadurch wird die Distanz, welche die Agenten zurücklegen müssen, ehe sie ihre eigentlichen Aufgaben ausführen können, minimiert. Allerdings hängen diese Koordinationsprozesse von dem Positions-Service der Umgebung ab. Wenn dieser Service ausfällt, sind die Koordinationsprozesse nicht mehr in der Lage, den nächstgelegenen Agenten zu bestimmen und funktionieren dementsprechend nicht mehr. In so einem Fall würde die Anwendung von der Fähigkeit zur strukturellen Adaption profitieren. Das Ziel einer solchen Adaption wäre der

Austausch der defekten Koordinationsprozesse, die von dem Positions-Service abhängen, durch die zufallsgesteuerten Koordinationsprozesse. Die Agenten haben hierbei allerdings keine direkte Information darüber, ob der Positions-Service der Umgebung ausgefallen ist oder nicht. Allerdings können sie die Zeit messen, die vergangen ist, seitdem sie das letzte Mal Koordinationsinformationen erhalten haben. Wenn sie innerhalb eines gewissen Zeitfensters keine Nachrichten mehr erhalten haben, können sie also annehmen, dass ein Fehlerfall eingetreten ist. Im Rahmen dieser Anwendungsstudie wartet ein Agent $20$ Sekunden, ehe er von einem solchen Fehler ausgeht und eine strukturelle Adaption vorschlägt.

Natürlich ist es dabei möglich, dass die Agenten aus anderen Gründen keine Koordinationsinformationen erhalten haben. Dieses kann z.B. dann der Fall sein, wenn die Agenten einen Teil der Umgebung erkunden, in dem sich keine anderen Agenten befinden. In diesem Fall wären die Agenten zu weit von den sendenden Agenten entfernt, um von den Koordinationsprozessen als nächstgelegener Agent ausgewählt zu werden. Daher ist es wichtig, dass die Agenten einen gemeinsamen Konsens darüber finden, ob die Koordinationsprozesse angepasst werden sollen, um lokale Phänomene zu überwinden.

Deswegen wird der in Abschnitt 4.4.4 beschriebene Abstimmungs-Algorithmus verwendet, um einen solchen Konsens über eine vorgeschlagene Adaption zu finden. Agenten, die innerhalb der letzten $20$ Sekunden keine Koordinationsinformationen erhalten haben, starten den Prozess und agieren als Abstimmungsleiter. Agenten, denen die strukturelle Adaption vorgeschlagen wird, prüfen, ob sie innerhalb der letzten $15$ Sekunden Koordinationsinformationen erhalten haben (Schwellwert). Wenn nicht, dann stimmen sie der vorgeschlagenen Adaption zu. Wie bereits erwähnt, kann der Schwellwert verwendet werden, um die Tendenz eines negativen Trends abzubilden, sodass Agenten einer vorgeschlagenen Adaption zustimmen können, bevor sie selber direkt dem fehlerhaften Verhalten ausgesetzt sind. Daher interpretieren sie hier einen potentiellen negativen Trend, sobald sie mit der vorgeschlagenen Adaption darüber informiert werden, dass ein anderer Agent bereits einem fehlerhaften Verhalten ausgesetzt ist. Natürlich sind dabei auch komplexere Bedingungen möglich als das Zeitfenster, welches aus Gründen der Anschaulichkeit in diesem Beispiel gewählt wurde.

Der Agent, der den Abstimmungsprozess initiiert hat, wartet, bis er alle Abstimmungsergebnisse erhalten hat (als Vereinfachung wurde in diesem Szenario auf einen Timeout verzichtet und der Verlust von Nachrichten vernachlässigt) und wertet sie dann aus. Wenn die benötigte Mehrheit von $75\%$ erreicht wurde, wird die strukturelle Adaption durchgeführt. Zum Start der Anwendung wird die Adaptionskomponente für $30$ Sekunden blockiert, da aufgrund des Aufbaus des Szenarios es in etwa diese Zeit dauert, bis erste potentielle Erzvorkommen entdeckt werden. Daher kann es zu Oszillierungsproblemen kommen, wenn sich das System zu früh auf eine Adaption einigt.

Quelltext 7.2 zeigt die deklarative Beschreibung der strukturellen Adaption. Der Agent, der den Abstimmungsprozess initiiert, erwartet $16$ Antworten (inkl.

einer von sich selbst) und eine Mehrheit von $75\%$ wird benötigt, damit die Adaption ausgeführt wird. Der `startDelay` gibt an, dass die *Adaptation Enactments* für $30$ Sekunden zur Startzeit blockiert werden. Das `reset` Flag gibt an, dass der Wert der entsprechenden Bedingung im Falle einer abgelehnten Abstimmung nicht zurückgesetzt werden soll. Das `single` Flag gibt an, dass die Adaption nur einmal ausgeführt werden soll. Im `realizations` Tag der XML-Beschreibung wird spezifiziert, welche Koordinationsprozesse im Rahmen der Adaption aktiviert bzw. deaktiviert werden sollen. Die `constraints` beschreiben, welche Werte in den betroffenen Agenten überwacht werden sollen, in diesem Fall handelt es sich um den BDI-Belief `no_msg_received`, der angibt, wie viel Zeit vergangen ist, seitdem zuletzt eine Koordinationsinformation empfangen wurde. Die `condition` und `threshold` Werte bilden in diesem Beispiel einfache Integer Werte ab. Geeignete Werte, sowohl für den `startDelay` als für die `condition` und `threshold` Werte, wurden im Rahmen eines iterativen Entwicklungsprozesses bestimmt (vgl. Abschnitt 4.4.1). Basierend auf einer Simulations-basierten Validierung sowohl des selbstorganisierenden Verhaltens als auch der strukturellen Adaption, wurden diese Werte entsprechend verfeinert, bis das System das gewünschte Verhalten zeigt.

```
1  <struct_adapt>
2  <agents>
3    <agent id="Producer"/>
4    <agent id="Carry"/>
5    <agent id="Sentry"/>
6  </agents>
7  <adaptations>
8   <adaptation id="change-to-random" answers="16" quorum="0.75" startDelay="
        30000" reset="false" single="true">
9    <realizations>
10    <realization id="latest_target_seen_nearest" activate="false"/>
11    <realization id="latest_target_analyzed_nearest" activate="false"/>
12    <realization id="latest_target_produced_nearest" activate="false"/>
13    <realization id="latest_target_seen_random" activate="true"/>
14    <realization id="latest_target_analyzed_random" activate="true"/>
15    <realization id="latest_target_produced_random" activate="true"/>
16   </realizations>
17   <constraints>
18    <constraint agent_id="Producer" element="no_msg_received" type="
          BDI_BELIEF" condition="20" threshold="15"/>
19    <constraint agent_id="Carry" element="no_msg_received" type="BDI_BELIEF"
          condition="20" threshold="15"/>
20    <constraint agent_id="Sentry" element="no_msg_received" type="BDI_BELIEF"
           condition="20" threshold="15"/>
21   </constraints>
22  </adaptation>
23 </adaptations>
24 </struct_adapt>
```

**Quelltext 7.2:** XML Quellcode der deklarativen Beschreibung der strukturellen Adaption für das MarsWorld Beispiel.

### Szenariobeschreibung

Das folgende Szenario wurde entworfen, um die Unterschiede zwischen den selbstorganisierenden und den auf einer zufälligen Auswahl basierenden Ko-

ordinationsprozessen sowie der Auswirkung der strukturellen Adaption zu messen und auszuwerten. Abbildung 7.1 zeigt das zu diesem Zweck entworfene Szenario: Es ist durch eine Dimension von *2 x 2* Längeneinheiten charakterisiert und enthält *20 Erzvorkommen* mit einer Gesamtmenge von *1000 Erzeinheiten*. In jedem der *4 Erz-Cluster* liegt dabei eine Gesamtmenge von *250 Erzeinheiten*. Die Erzvorkommen in der linken, oberen Ecke werden als Cluster 1 bezeichnet, die rechte, obere Ecke als Cluster 2, die linke untere Ecke als Cluster 3 und die rechte, untere Ecke schließlich als Cluster 4. Das Szenario wurde in dieser Form konzipiert, um das Potential der selbstorganisierenden Koordinationsprozesse, die auf dem Nachbarschafts-Ansatz basieren und die Formierung von lokalen Abbaugruppen fördern, aufzuzeigen. Die Heimatbasis befindet sich in der Mitte der Umgebung. Von hier aus beginnen alle Agenten zum Start der Anwendung zufällig die Umgebung zu erkunden. Wenn ein *Sentry* Agent auf ein Erzvorkommen stößt, beginnt er es zu analysieren, während die *Producer* und *Carry* Agenten über den aktiven Koordinationsprozess einen *Sentry* Agenten informieren, wenn sie ihrerseits auf ein Erzvorkommen gestoßen sind. Das Szenario ist beendet, wenn das gesamte Erz abgebaut und zur Heimatbasis transportiert wurde. Die Anzahl der teilnehmenden Agenten und deren Konfiguration ist in Tabelle 7.1 aufgelistet.

**Tabelle 7.1.:** Konfiguration der an der MarsWorld teilnehmenden Agenten.

| Agententyp | Anzahl | Sichtradius | Geschwindigkeit | Ladekapazität (Erz) |
|---|---|---|---|---|
| Sentry | 4 | 0,15 | 0,1 | - |
| Producer | 4 | 0,1 | 0,1 | - |
| Carry | 8 | 0,05 | 0,2 | 20 |

Abbildung 7.2 zeigt die laufende MarsWorld Anwendung. Nachdem 59 Sekunden vergangen sind, haben die Agenten bereits 150 Erzeinheiten abgebaut. Das Szenario wurde dabei mit den auf dem Nachbarschafts-Ansatz basierenden Koordinationsprozessen ausgeführt und es scheinen sich drei lokale Abbaugruppen herausgebildet zu haben, die die Erz-Cluster 2, 3 und 4 abbauen.

### Auswertung

Das *MarsWorld* Szenario wurde auf Basis der Jadex-Multiagentenplattform [BP12c] realisiert. Die drei Agententypen, *Sentry*, *Producer* und *Carry* wurden dabei als BDI-Agenten entworfen. Um die Unterschiede zwischen dem selbstorganisierenden und den auf einer zufälligen Auswahl basierenden Koordinationsprozessen sowie die Auswirkungen der strukturellen Adaption zu messen und auszuwerten, wurde das Szenario für jede der beiden Koordinationsprozessausprägungen sowie für ein Szenario mit der beschriebenen strukturellen Adaption jeweils 50 Mal ausgeführt und die Ergebnisse gemittelt. Die automatisierte Ausführung dieser Szenarien erfolgte mit Hilfe eines Management-Frameworks für Multiagenten-basierte Simulationen

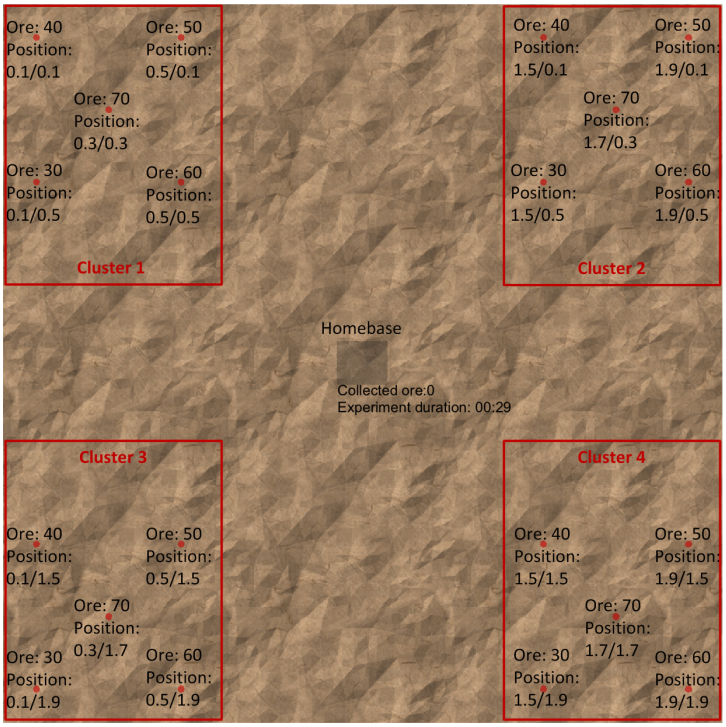



**Abbildung 7.1.:** Screenshot des MarsWorld Szenarios mit hervorgehobenen Erz-Clustern - aus (PRD15, S. 9).

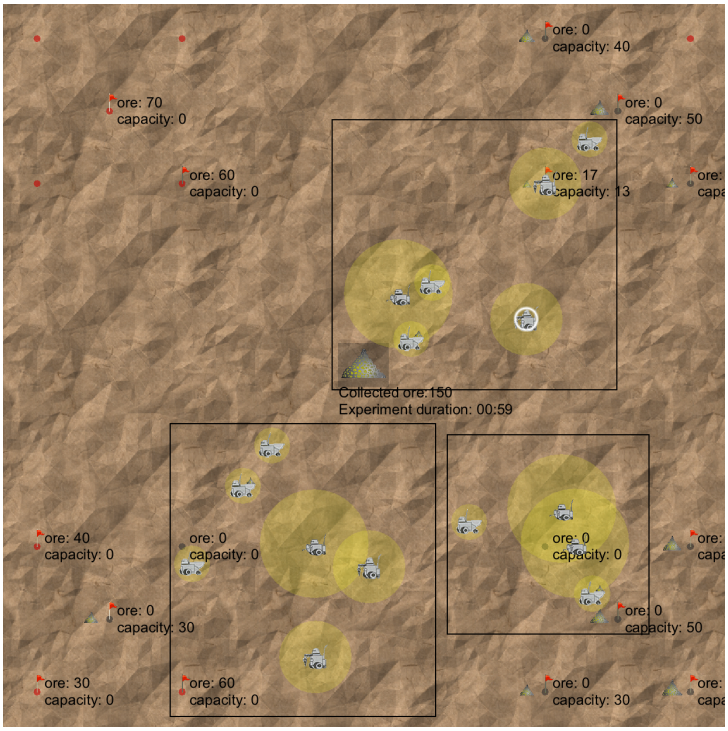



**Abbildung 7.2.:** Screenshot der laufenden MarsWorld Anwendung mit hervorgehobenen, selbstorganisierend geformten lokalen Gruppen - aus (PRD15, S. 10).

[VL11] und eines Evaluationsframeworks für die Überwachung und Auswertung von vordefinierten Systemobservablen [VL12]. Beide wurden im Rahmen der Dissertation von *Vilenica* [Vil13] entwickelt, um die automatisierte Simulation und Evaluation von selbstorganisierenden Systemen zu ermöglichen.

Die sechs unterschiedlichen Koordinationsprozesse (drei pro Ausprägung) wurden, wie bereits erwähnt, mit der deklarativen *MASDynamics* Beschreibungssprache modelliert. Für die technische Realisierung der Koordinationsmedien wurden zwei Koordinationsmechanismen implementiert. Der erste wählt jeweils zufällig einen Agenten vom passenden Typ aus. Dieser Mechanismus wurde von den drei Koordinationsprozessen, die auf der zufälligen Auswahl beruhen, verwendet. Der zweite Koordinationsmechanismus verwendete den Positions-Service der Umgebung, um den jeweils nächstgelegenen Agenten vom passenden Typ zu selektieren. Dieser Mechanismus wurde entsprechend von drei Koordinationsprozessen verwendet, die auf dem Nachbarschafts-Ansatz basieren.

**Koordination** Um den Einfluss der Koordinationsprozesse auf die Geschwindigkeit des Erzabbau zu messen, wurden $50$ Simulationsläufe durchgeführt und dabei die Menge an analysiertem, abgebautem und eingesammeltem Erz gemessen. Dabei wird deutlich, dass die selbstorganisierenden Prozesse einen Performanzvorteil gegenüber den anderen Prozessen erzielen, da bei gleicher Laufzeit jeweils mehr Erz analysiert, produziert und schließlich eingesammelt wurde. In Abbildung 7.3 sind die gemittelten Werte über die Simulationsläufe aufgetragen. Die Resultate der selbstorganisierenden Prozesse, die auf dem Nachbarschafts-Ansatz basieren, sind in der Abbildung mit dem *SO*-Präfix gekennzeichnet und die Prozesse, die auf der zufälligen Auswahl basieren, mit dem *NonSO*-Präfix.

Eine weiterführende Analyse des Vorteils der selbstorganisierenden Koordinationsprozesse gegenüber ihren nicht selbstorganisierenden Gegenstücken ist in Abbildung 7.4 dargestellt. Dabei wird zunächst deutlich, dass die selbstorganisierenden Koordinationsprozesse aufgrund der emergenten Herausbildung von lokalen Abbaugruppen zu einem effektiveren Abbau von Erz führen. Weiterhin zeigt sich hierbei, dass der Einfluss des selbstorganisierenden Verhaltens auf die Menge an analysierten Erzvorkommen deutlich geringer ist als der Einfluss auf die Menge an abgebauten und eingesammelten Erz. Das lässt sich damit begründen, dass die *Sentry* Agenten von sich aus in der Lage sind, potentielle Erzvorkommen aufzuspüren und zu analysieren. Daher sind die *Sentry* Agenten nicht zwingend von den Koordinationsinformationen abhängig, die sie von den *Producer* und *Carry* Agenten erhalten. Die *Producer* und *Carry* Agenten hingegen sind nur in der Lage, Erz abzubauen bzw. es abzutransportieren, wenn sie Koordinationsinformationen über entsprechende Vorkommen erhalten. Daher hängt ihre Fähigkeit, Erz abzubauen oder abzutransportieren, stark von der Koordination ab. Die jeweils zeitlich versetzt abnehmenden Effizienzraten in Abbildung 7.4 hängen mit den jeweils ab-

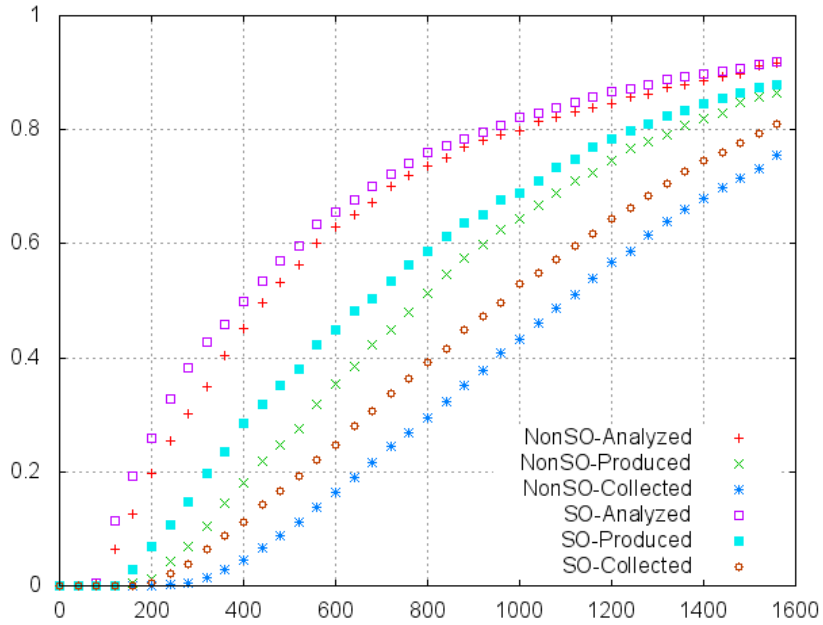



**Abbildung 7.3.:** Evaluation der Koordinationsprozesse im MarsWorld Szenario aus (PRD15, S. 10). Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse gibt Auskunft über den prozentualen Anteil an analysierten (*analyzed*), produzierten (*produced*) und eingesammelten (*collected*) Erz.

nehmenden Mengen an Erz zusammen, das noch nicht analysiert, abgebaut bzw. abtransportiert wurde.

**Strukturelle Adaption** Um den Einfluss der strukturellen Adaption zu untersuchen, wurden erneut 50 Simulationen ausgeführt und die Ergebnisse gemittelt. In jedem der Szenarien waren zunächst die Koordinationsprozesse aktiv, die auf dem Nachbarschafts-Ansatz basierten. Nach 60 Sekunden wurde jeweils ein Fehler im Positions-Service der Umgebung simuliert. Damit waren die entsprechenden Koordinationsmechanismen ab diesem Zeitpunkt nicht mehr in der Lage, den nächstgelegenen Agenten zu bestimmen und konnten somit keine Koordinationsinformationen mehr verteilen. Wie zuvor beschrieben, werden die *Adaptation Enactments* für die ersten 30 Sekunden blockiert, um Oszillationsprobleme zu verhindern, die auftreten, da die Agenten zunächst zufällig die Umgebung erkunden und daher einige Zeit vergeht, ehe sie auf erste potentielle Erzvorkommen stoßen und somit erst dann Koordinationsinformationen verschickt werden. Die Ergebnisse der Analyse der strukturellen Adaption sind in Abbildung 7.5 dargestellt.

Zunächst zeigt die Abbildung, wie der simulierte Fehler nach 60 Sekunden auftritt (die Abweichungen liegen hierbei an Ungenauigkeiten im Zeitdiensts der genutzten Agentenplattform). Erste Abstimmungsversuche für eine strukturelle Adaption können nach 50 Sekunden beobachtet werden. Diese Zahl erklärt sich dadurch, dass die Adaptionskomponenten für die ersten 30 Sekun-

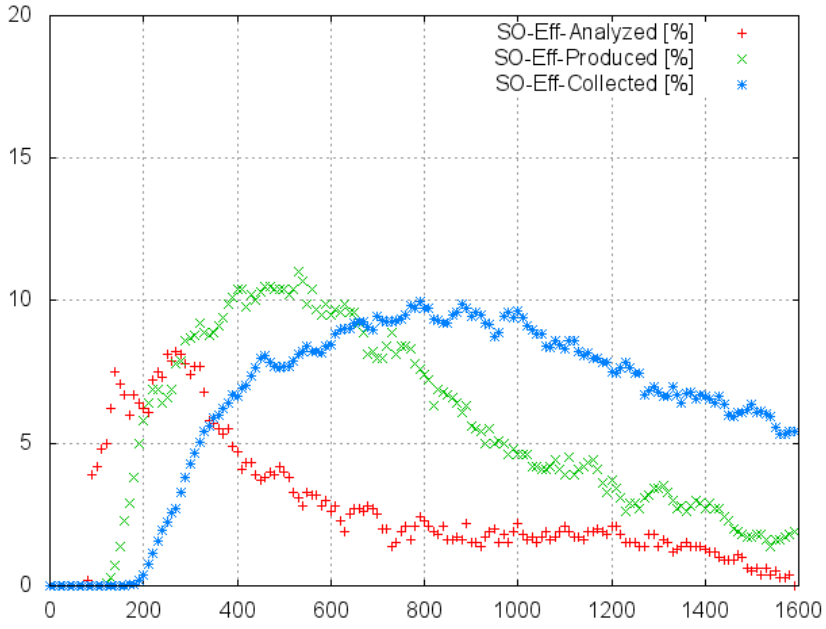



**Abbildung 7.4.:** Effizienz der selbstorganisierenden Prozesse im MarsWorld Szenario aus (PRD15, S. 11). Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse zeigt prozentual, wie viel effektiver die selbstorganisierenden Koordinationsprozesse gegenüber den anderen Koordinationsprozessen sind.

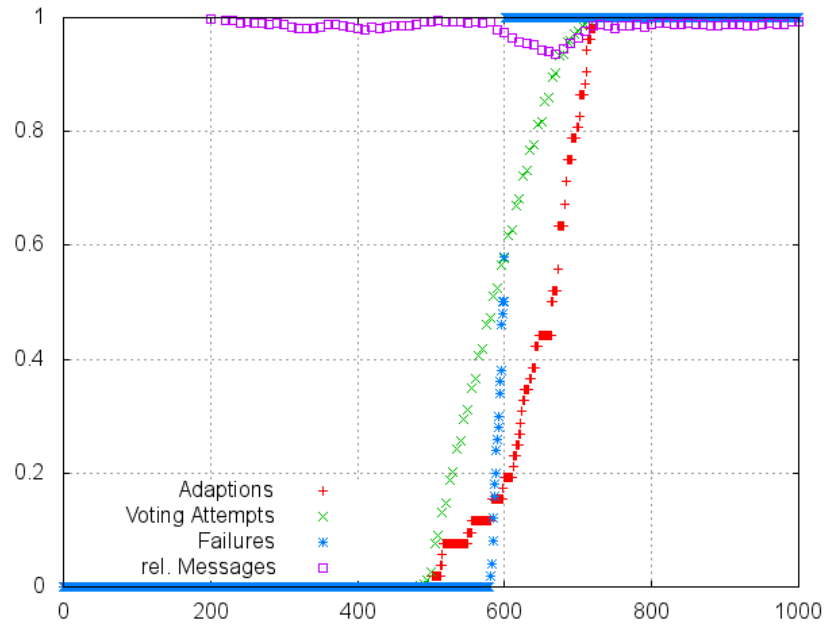



**Abbildung 7.5.:** Auswertung der strukturellen Adaptionen im MarsWorld Szenario aus (PRD15, S. 11). Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse zeigt die prozentuale Anzahl der gestarteten Abstimmungsprozesse (*Voting Attempts*), den prozentualen Anteil an simulierten Fehlern im Positionsservice (*Failures*), die prozentuale Anzahl an durchgeführten Adaptionen (*Adaptions*) und die relative Anzahl von Koordinationsnachrichten aus den Szenarien mit simulierten Fehler in Relation zur Anzahl von Koordinationsnachrichten aus einem Szenario ohne simulierten Fehlerfall (*rel. Messages*).

den blockiert sind und weitere 20 Sekunden vergehen müssen, ehe die Adaptionsbedingung erfüllt sein kann. Es zeigt sich, dass die Anzahl der Adaptionsversuche ansteigt, bis im Mittel alle Systeme die strukturelle Adaption durchgeführt haben. In diesem Fall war die Adaption als *unique* (einzigartig) deklariert, das bedeutet, dass nach erfolgreicher Adaption keine weiteren Adaptionsversuche gestartet werden. Es wird ersichtlich, dass in etwa 15% der Fälle das System seine Struktur adaptiert, bevor der Fehler eintrat. Wie zuvor beschrieben, wurde eine Mehrheit von 75% vorausgesetzt, damit die Adaption durchgeführt wird. Damit ein Abstimmungsprozess initiiert wird, muss mindestens ein Agent innerhalb von 20 Sekunden keine Koordinationsinformationen erhalten haben. Um dann eine Mehrheit für die Adaption zu erzielen, müssen mindestens 11 andere Agenten innerhalb der letzten 15 Sekunden (der entsprechende Schwellwert der Adaption) keine Koordinationsinformationen erhalten haben. Aufgrund der zufälligen Exploration der Umgebung kann dieses Phänomen auftreten, bevor der eigentliche Fehlerfall simuliert wurde, wenn zu diesem Zeitpunkt keine oder nur wenige potentielle Erzvorkommen von den Agenten gefunden wurden. Die Agenten interpretieren in diesem Fall die Abwesenheit von Koordinationsinformationen als einen potentiellen Fehler in den entsprechenden Koordinationsprozessen und stimmen daher einer vorgeschlagenen Adaption zu.

Ein längerer Blockierungszeitraum für die *Adaptation Enactments* zur Startzeit oder strengere Konditions- und Schwellwerte für die Adaption können zu weniger solcher verfrühten Adaptionen führen. Andererseits würde sich dadurch auch die Reaktionszeit auf einen tatsächlichen Fehlerfall erhören. Wie bereits in Abschnitt 4.4.1 beschrieben, ist das Identifizieren von geeigneten Adaptionsparametern eine der Hauptherausforderungen bei der Umsetzung von strukturellen Adaptionen der Koordinationsprozesse und sollte daher Teil eines iterativen, simulationsgetriebenen Entwicklungsansatzes sein, der mehrere Durchläufe erfordern kann, bis geeignete Parameter identifiziert werden können.

Weiterhin ist in der Grafik die relative Anzahl von Koordinationsnachrichten aus den Szenarien mit simuliertem Fehler in Relation zur Anzahl von Koordinationsnachrichten aus einem Szenario ohne simulierten Fehlerfall dargestellt (*rel. Messages*). Die Linie zeigt dabei, dass es zunächst keine Abweichung zwischen diesen beiden Szenarien gibt, bis der Fehlerfall auftritt. Dann sinkt die relative Anzahl der Nachrichten etwas ab, ehe sich der Wert nach der strukturellen Adaption wieder erholt und schließlich wieder das Level des fehlerfreien Szenarios erreicht. Damit wird gezeigt, wie die strukturelle Adaption in der Lage ist, das fehlerhafte Verhalten, das durch einen externen Fehler im Positions-Service der Umgebung verursacht wurde, zu reparieren.

### Zusammenfassung und Bewertung

In der Anwendungsstudie, die im Rahmen von [PRD15] durchgeführt wurde, wurde anhand des *MarsWorld* Szenarios das Konzept der strukturellen Adap-

tion von Koordinationsprozessen veranschaulicht. Dabei wurde gezeigt, wie automatisiert zur Laufzeit die Struktur der selbstorganisierenden Dynamik eines Systems angepasst werden kann, indem ein zur Designzeit antizipierter Austausch der Koordinationsprozesse, welche die vorgesehene selbstorganisierende Dynamik abbilden, erfolgt. Hierbei wurde sowohl auf die technische Realisierung dieses Adaptionsprozesses eingegangen als auch auf den Entwicklungsansatz, der diesen antizipierten Adaptionen zugrunde liegt. Damit konnte im Sinne einer Validierung gezeigt werden, dass das Konzept zur strukturellen Adaption von Koordinationsprozessen geeignet ist, um dynamische Ineffizienzen in einem selbstorganisierenden System zu überwinden.

### 7.1.2. Bike-Sharing Washington, D.C.

Um die Verwendung des in den Abschnitt 6.2 präsentierten dezentralen Koordinationsframework (DeCoF) zur Realisierung von selbstorganisierenden Verhalten zu veranschaulichen, wird an dieser Stelle das bereits eingangs in Abschnitt 2.5.2 präsentierte Beispiel des selbstorganisierenden Fahrradverleihsystems aufgegriffen. Wie bereits in Abschnitt 2.5.2 erwähnt, hängt der Betrieb von modernen Fahrradverleihsystemen in großen Städten primär von der *Verfügbarkeit* von Fahrräder an den Ausleihstationen ab. Dabei sind die Systeme mit dem Problem der *Zuverlässigkeit* konfrontiert, bei dem gerade in Berufsverkehr-Situationen einige Stationen leerlaufen, während andere wiederum volllaufen. In dem Abschnitt wurde daher ein selbstorganisierender Ansatz vorgestellt, der die Umverteilung von Fahrrädern in Form von LKW-Transporten, wie sie von den meisten Betreibern durchgeführt werden, ergänzen kann. Der Ansatz basiert dabei auf dem Gedanken, die Benutzer des Systems durch geeignete Anreize dazu zu ermutigen, bei der Ausleihe und der Rückgabe von Fahrräder auch in der Nähe befindliche Stationen in Betracht zu ziehen, um dadurch in selbstorganisierender Weise für eine erhöhte *Verfügbarkeit* von Fahrräder an den Ausleihstationen zu sorgen. Der Ausleihvorgang aus der Perspektive eines kooperativen Benutzers sieht dabei wie folgt aus: Zunächst nähert sich der Benutzer einer Ausleihstation an, um dort ein Fahrrad auszuleihen. Wenn an dieser Station keine oder nur noch wenige Fahrräder verfügbar sind, wird ihm eine alternative Ausleihstation in der Nähe vorgeschlagen, die noch über ausreichend Fahrräder verfügt. Nun kann der Benutzer entscheiden, ob er dem Vorschlag folgt oder nicht. Wenn ein Benutzer ein Fahrrad an einer Station abgeben will, wird ihm analog zum Ausleihvorgang eine alternative Rückgabestation empfohlen, sofern die eigentliche Rückgabestation bereits voll oder zumindest schon sehr stark ausgelastet ist (siehe auch Abbildung 2.15 für eine grafische Darstellung dieses Vorgangs).

Der Erfolg dieses Ansatzes hängt primär von zwei Faktoren ab. Zunächst muss ein geeignetes Anreizsystem gefunden werden, das die Bereitschaft der Benutzer erhöht, die notwendigen Umwege in Kauf zu nehmen. Solche Anreize können z.B. in Form von Belohnungen, wie Freiminuten für die nächste Ausleihe oder monetäre Entschädigungen, umgesetzt werden. Die Wahl eines

geeigneten Anreizsystems liegt allerdings nicht im Fokus dieser Arbeit, sodass an dieser Stelle angenommen wird, dass von den Betreibern ein geeigneter Ansatz gefunden wird.

Dieser Abschnitt fokussiert viel mehr den dezentralen Koordinationsansatz, der zur Bestimmung der alternativen Ausleih- und Abgabestationen, die den Benutzern vorgeschlagen werden, verwendet wird. Im Folgenden wird daher zunächst der dezentrale Koordinationsansatz konzeptionell beschrieben, ehe die technische Umsetzung des Ansatzes mit *DeCoF* für den agentenbasierten Logistik-Simulator *RinSim* [vLH12] beschrieben wird. *RinSim* wurde verwendet, um eine Simulation vom Fahrradverleihsystem in Washington, D.C. zu realisieren. Dabei sind sowohl die Benutzer, die die Fahrräder ausleihen, als auch Ausleihstationen jeweils als Software-Agenten realisiert. Basierend auf einer Analyse der Historie-Daten über die Ausleihvorgänge in Washington, D.C. im Jahr 2014 wurde dabei ein idealisierter Montag auf Basis von stochastischen Wahrscheinlichkeiten erstellt. Das daraus resultierende Modell simuliert auf Stundenbasis. Für jede Stunde wurden basierend auf den Historie-Daten über die Ausleihvorgänge Abfahrts- und davon abhängige Ankunftswahrscheinlichkeiten für die einzelnen Stationen berechnet. Um den unterschiedlichen Andrang an den verschiedenen Stationen zu unterschiedlichen Zeiten des Tages zu simulieren, wurde ein Mittelwert für jede Stunde basierend auf dem tatsächlichen Andrang gemäß den Historie-Daten errechnet. Ausgehend von diesem Modell werden während der Ausführung der Simulation entsprechend dem mittleren Andrang für den aktuellen Zeitabschnitt gleichmäßig verteilt Agenten gestartet, die die Benutzer abbilden. Basierend auf den Abfahrts- und Ankunftswahrscheinlichkeiten wird dabei bestimmt, von welcher Station zu welcher anderen Station die Agenten fahren. Dabei bewegen sie sich mit einer konstanten Geschwindigkeit entlang der Straßen, die auf einem Auszug aus der *OpenStreetMap* Karte [HW08] von Washington, D.C. basieren und innerhalb der Simulation als Graph abgebildet sind. Das Gewicht der Kanten repräsentiert dabei die Entfernung zum nächsten Knotenpunkt. Für die Simulation bewegen sich die Fahrradfahrer mit einer konstanten Geschwindigkeit von $18\frac{km}{h}$ und alle Stationen verfügen über $20$ Plätze, von denen initial $10$ mit Fahrrädern belegt sind. Innerhalb der Simulation ist eine Überbelegung der Stationen erlaubt, d.h. ein Benutzer kann ein Fahrrad auch dann an einer Station abgegeben, wenn diese eigentlich keine freien Plätze mehr hat.

### Dezentrale Koordination

Um die alternativen Ausleih- und Rückgabestationen, die den Benutzern vorgeschlagen werden, zu bestimmen, wurde ein dezentraler Koordinationsansatz verwendet und mit Hilfe von *DeCoF* für das Simulationssystem umgesetzt. Dabei senden die Stationen periodisch ihre aktuelle Auslastungsrate an alle anderen Stationen, die innerhalb eines gewissen Kommunikationsradius liegen. Der Kommunikationsradius ist dabei ein konfigurierbarer Koordina-

tionsparameter. Die Auslastungsrate $\mu$ ergibt sich dabei aus der Anzahl der belegten Plätze $P_b$ geteilt durch die Anzahl der freien Plätze $P_f$, also $\mu = \frac{P_b}{P_f}$. Stationen, die solche Statusupdates erhalten, sammeln diese ein und verwendeten sie, um die alternativen Ausleih- und Rückgabestationen zu berechnen. Diese Berechnung erfolgt also dezentral. Bei der Berechnung bestimmt die Station aus allen Statusupdates zunächst die beiden Stationen mit der niedrigsten und höchsten Auslastungsrate. Die Station mit der niedrigsten Auslastungsrate wird dann einem Benutzer als Alternative vorgeschlagen, wenn er sein Fahrrad an der Station abgeben möchte, die Station selber aber schon sehr stark ausgelastet ist. Analog dazu wird die Station mit der höchsten Auslastungsrate dem Benutzer als Alternative vorgeschlagen, wenn er ein Fahrrad an der Station ausleihen möchte, die Station aber leer ist oder über nur noch sehr wenige Fahrräder verfügt. Im Rahmen der Untersuchung der Effektivität dieser Strategie in [PDR15a] und [PDR16b] wurden dabei den Benutzern Alternativen vorgeschlagen, sobald eine Station im Fall einer Ausleihe eine Auslastungsrate von weniger als $25\%$ und im Falle einer Abgabe eine Auslastungsrate von mehr als $75\%$ aufweist.

Die Stationen wurde im Rahmen der Simulation als *RinSim*-spezifische Agenten umgesetzt. Dabei verwaltet jeder Stations-Agent die aktuell an dieser Station befindlichen Fahrräder und stellt jeweils eine Methode zum Ausleihen (`borrowBike() : Bike`) und zum Zurückgeben (`returnBike(Bike bike)`) von Fahrrädern zu Verfügung. Innerhalb dieser Methoden werden den Benutzern die alternativen Ausleih- bzw. Rückgabestationen vorgeschlagen. Die Benutzer entscheiden sich dann anhand ihrer Kooperationsrate, die ein variabler Parameter der Simulation ist, mit einer bestimmten Wahrscheinlichkeit dafür, ob sie die vorgeschlagene Alternative akzeptieren. Basierend auf dem Integrationsansatz vom *DeCoF* (siehe Abschnitt 6.2.2), müssen die Stations-Agenten das `ICoordinatable` Interface implementieren und mit einem Koordinationsendpunkt ausgestattet werden, um die dezentralen Koordinationsprozesse zu realisieren, die zur Bestimmung der alternativen Stationen benötigt werden.

Um eine Komponente mit einem Koordinationsendpunkt auszustatten und diesen über einen Eventbus lose gekoppelt mit der Komponente zu verbinden, wird vom Framework eine statische Hilfsmethode zur Verfügung gestellt: `CoordinationHelper.equipCoordination(ICoordinatable comp)`.
Diese Methode erzeugt einen Koordinationsendpunkt und den entsprechenden Eventbus, über den der Endpunkt über Koordinationsereignisse innerhalb der Komponente informiert wird. Der Endpunkt registriert sich dabei bei dem Eventbus und dieser wird von der Methode zurückgegeben. Innerhalb der Komponente muss nun eine lokale Referenz auf den Eventbus gespeichert werden, sodass dieser von der Komponente verwendet werden kann, um einen (eventuellen) Endpunkt über Koordinationsereignisse zu informieren. Im Rahmen der RinSim-basierten Simulation verwenden die Stations-Agenten den Eventbus, um in jeder Minute der Simulation einen sogenannten `BikeStationStatusUpdateEvent` als Ko-

ordinationsereignis, das ihre aktuelle Auslastungsrate enthält, auszulösen: `coordEventBus.post(new BikeStationStatusUpdateEvent(...))`. Vom zugehörigen Endpunkt wird dieses Ereignis aufgenommen und als Koordinationsinformation über das entsprechende Koordinationsmedium verbreitet. In diesem Fall erweitert das `RoadBasedCoordinationMedium` die abstrakte `CoordinationMedium` Oberklasse mit RinSim-spezifischer Koordinationslogik. Dazu bekommt das `RoadBasedCoordinationMedium` eine Referenz auf das Straßenmodell des Simulators, damit es Zugriff auf die Umgebung der Simulation und damit auf die Positionen der Ausleihstationen bekommt. Mit Hilfe des Kommunikationsradius, der ein konfigurierbarer Koordinationsparameter des Koordinationsmediums ist, kann vom Medium anhand dieser Informationen die Untermenge der Stationen selektiert werden, die die Koordinationsinformation erhalten sollen. Dabei wird die Position der emittierenden Station als Ausgangspunkt verwendet, von dem aus der Kommunikationsradius die Auswahl begrenzt. Die Koordinationsendpunkte der selektierten Stationen erhalten über das Koordinationsmedium diese Koordinationsinformationen und lösen die weitere Verarbeitung der enthaltenen Koordinationsereignisse in den Stations-Agenten aus. Dazu wird die `handleCoordinationEvent(CoordinationEvent<?> ce)` Methode des `ICoordinatable` Interfaces aufgerufen. Hier erfolgt dann die Berechnung der alternativen Ausleih- und Rückgabestationen, die den Benutzern vorgeschlagen werden.

In Quelltext 7.3 ist die deklarative Koordinationsprozessbeschreibung dargestellt, die die Koordinationsendpunkte entsprechend instruiert und den Koordinationsprozess strukturiert. Hier wird gezeigt, wie die `Bikestation` Agenten auf die `bikestation` Rolle abgebildet werden und wie diese Rolle verwendet wird, um den dezentralen Koordinationsprozess zu instruieren. Die Beschreibung der Koordinationslinks (`deco-link`) enthält dabei Informationen über die betroffenen Rollen, in diesem Fall die `bikestation` Rolle, und wie die Koordinationsereignisse aufeinander abgebildet werden sollen. Ferner wird angegeben, welches Koordinationsmedium für die Informationsausbreitung verwendet werden soll. Im Fall dieses Beispiels wird das `RoadBasedCoordinationMedium` verwendet, bei dem über den `maxCommRange` Koordinationsparameter die Informationsausbreitung limitiert wird. In Fall des Quelltext wird ein Kommunikationsradius von $1,5$ km verwendet (siehe Zeile 14).

Abbildung 7.6 zeigt an dieser Stelle das Kommunikationsmodell, das aus dem Koordinationsansatz resultiert. Dabei ist für jede Ausleihstation der ringförmige Kommunikationsbereich eingezeichnet, der aus dem gewählten Kommunikationsradius (Koordinationsparameter) resultiert. Stationen können also nur mit anderen Stationen innerhalb ihres Kommunikationsradius Koordinationsinformationen auszutauschen. Das schränkt zwar einerseits die Ausbreitung der Koordinationsinformationen ein und limitiert somit die Auswahlmöglichkeiten innerhalb der dezentralen Bestimmung der alternativen Stationen, andererseits sorgt ein geringer Kommunikationsradi-

```
<coordination-description context="BikeSharing">
 <role-definitions>
  <role name="bikestation">
   <components>
    <component class="de.haw.c4das.bikesharing.simulation.Bikestation" />
   </components>
  </role>
 </role-definitions>
 <deco-link-definitions>
  <deco-link id="bs-update">
   <from role="bikestation" event="de.haw.c4das.bikesharing.simulation.
       coordination.BikeStationStatusUpdateEvent" />
   <medium class="de.haw.c4das.bikesharing.simulation.coordination.
       RoadBasedCoordinationMedium">
    <properties>
     <property key="maxCommRange" value="1.5" />
    </properties>
   </medium>
   <to role="bikestation" event="de.haw.c4das.bikesharing.simulation.
       coordination.BikeStationStatusUpdateEvent" />
  </deco-link>
 </deco-link-definitions>
</coordination-description>
```

**Quelltext 7.3:** XML-basierte Beschreibung des Koordinationsprozesses zur Verbreitung der Auslastungsraten im Bike-Sharing Beispiel.

us eben auch dafür, dass die Benutzer nur geringe Umwege im Fall einer Umleitung zurücklegen müssen.

Die selbstorganisierende Dynamik, die sich aus den zuvor beschriebenen Konzepten herausbildet, setzt sich aus zwei Aspekten zusammen: Der dezentralen Koordination der Ausleihstationen, um die alternativen Ausleih- und Rückgabestation zu bestimmen und der Kooperationsbereitschaft der Benutzer, einem solchen Vorschlag zu folgen. Ohne die Kooperationsbereitschaft der Benutzer wäre die dezentrale Koordination in diesem Fall effektlos, ebenso wie die Kooperationsbereitschaft der Benutzer nur zur gewünschten selbstorganisierenden Dynamik beitragen kann, wenn im Rahmen der Koordinationsprozesse die alternativen Stationen bestimmt werden. Abbildung 7.7 veranschaulicht diese Dynamik.

Um nun den Einfluss der selbstorganisierenden Dynamik in Abhängigkeit vom Kommunikationsradius als relevanten Koordinationsparameter zu evaluieren, wurden im Rahmen von [PDR16b] eine Reihe von unterschiedlichen Szenarien mit fluktuierenden Kommunikationsradien sowie ein Referenzszenario ohne selbstorganisierendes Verhalten auf Basis der Historie-Daten von Washington, D.C. (2014) simuliert. Dabei wurde für alle dieser Szenarien eine Kooperationsbereitschaft der Benutzer von $100\%$ angenommen. Das bedeutet, dass ein Benutzer immer einer vorgeschlagenen Umleitung zu einer alternativen Ausleih- oder Rückgabestation folgt. Diese Annahme wurde gemacht, um präzise den Einfluss des Kommunikationsradius als Koordinationsparameter bestimmten zu können. Für eine detaillierte Analyse bzgl. des Einflusses der Kooperationsbereitschaft der Benutzer auf das selbstorgani-

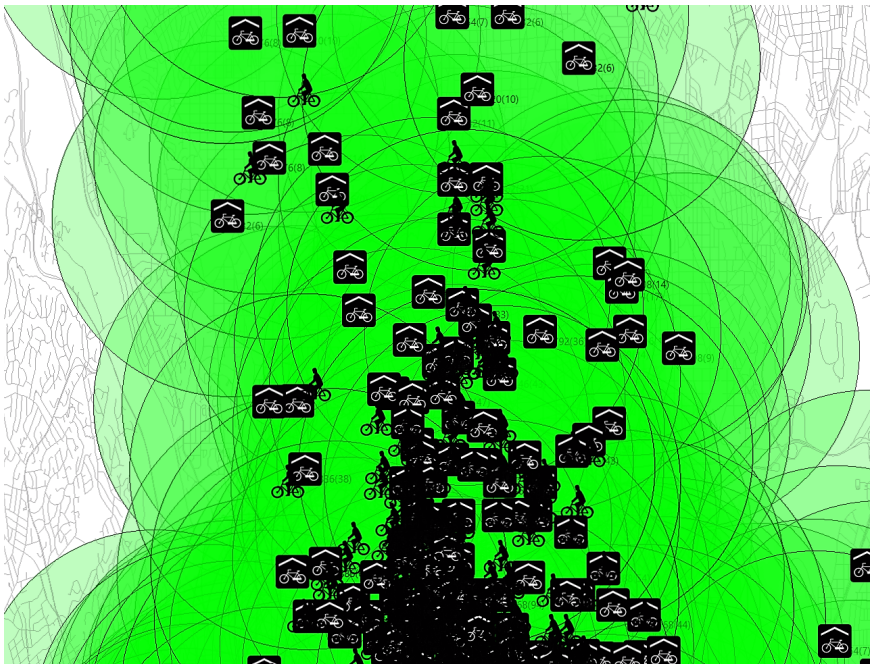


**Abbildung 7.6.:** Auszug der simulierten Karte für das Bike-Sharing Beispiel, hier sind einige Radfahrer sowie die Ausleihstationen und der gewählte Kommunikationsradius gezeigt - aus (PDR16b, S. 7).

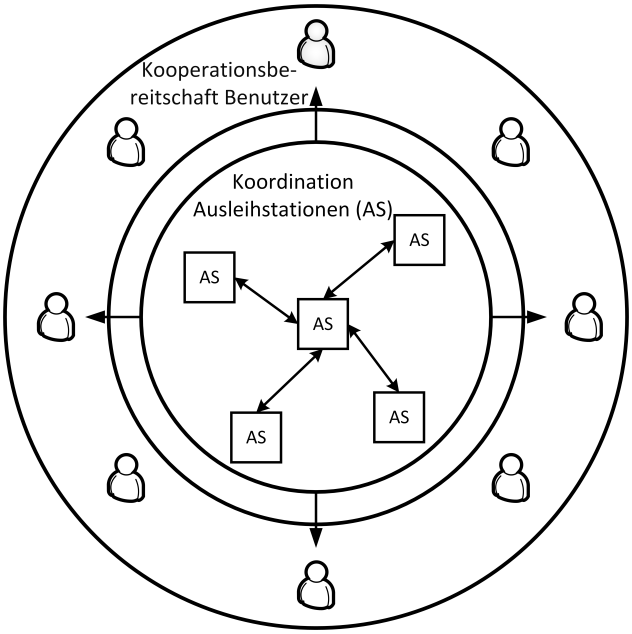



**Abbildung 7.7.:** Selbstorganisierende Dynamik des Fahrradverleihsystems - nach (PDR16b).

sierende Verhalten wird auf die Primärquellen in [PRV13, PDR15a] und die Zusammenfassung der dort erzielten Ergebnisse in Abschnitt 2.5.2 verwiesen.

Abbildung 7.8 zeigt zunächst die Resultate eines Simulationsszenarios ohne selbstorganisierendes Verhalten im Vergleich zu einem Szenario, bei dem der Kommunikationsradius auf $1,5$ km begrenzt ist. Die Abbildung zeigt dabei die Anzahl der Stationen, die in einem *normalen* Zustand, also weder voll noch leer sind und wie sich deren Verlauf über den Tag entwickelt. Zunächst lässt sich dabei in beiden Fällen beobachten, wie die Anzahl der normalen Stationen mit Beginn des morgendlichen Berufsverkehrs (etwa um Minute $420$) abnimmt. Im weiteren Verlauf des Tages fluktuieren diese Werte nur wenig. Im späteren Nachmittag (um Minute $1000$ herum) erholt sich die Anzahl der normalen Stationen wieder ein wenig. Dieses Verhalten kann durch die Bewegungen von Pendlern zu den Hauptverkehrszeiten erklärt werden. Diese leihen morgens Fahrräder in den Vororten aus und fahren damit in das Stadtzentrum, um nachmittags wieder zurück in die Vororte zu fahren. Stationen in den Vororten laufen daher während der morgendlichen Berufsverkehrszeit leer, während die Stationen im Stadtzentrum gefüllt werden. Die Pendler bewegen diese Fahrräder nun am Nachmittag von den überbelegten Stationen im Stadtzentrum zurück und füllen damit die Stationen in den Vororten wieder auf. Durch die selbstorganisierende Umverteilung wird die *Verfügbarkeit* der Fahrräder an den Stationen verbessert, so sinkt die Anzahl an normalen Stationen im Laufe des Tages nicht unter $320$, wohingegen ohne Umverteilung die Anzahl auf $280$ Stationen absinkt.

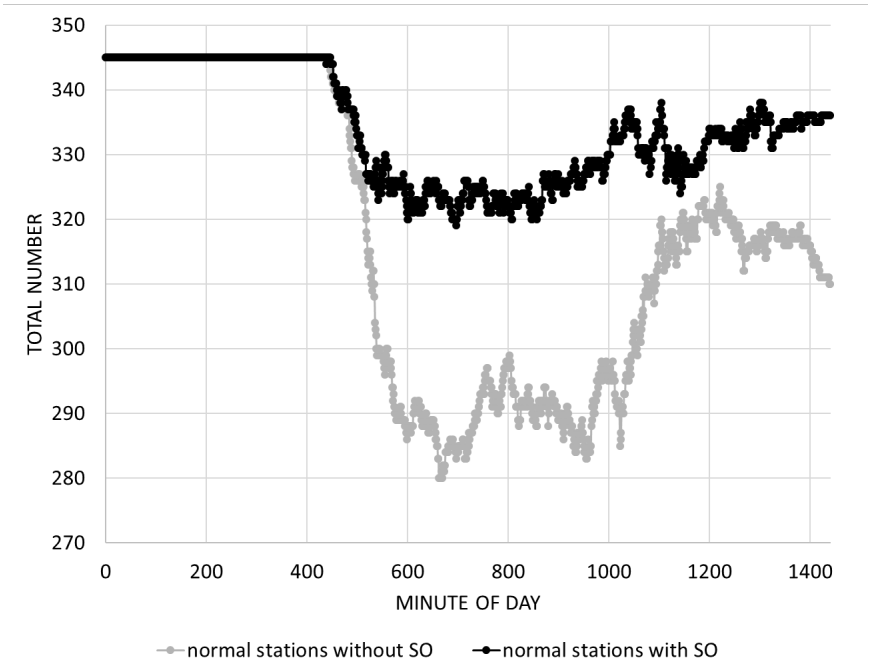



**Abbildung 7.8.:** Vergleich der Resultate für ein Simulationsszenario mit selbstorganisierendem Verhalten und einem ohne - aus (PDR16b, S. 8).

Um nun den Einfluss des Kommunikationsradius auf die selbstorganisierende Umverteilung der Fahrräder weiter zu untersuchen, wurden sechs Simulationsläufe mit unterschiedlichen Kommunikationsradien durchgeführt. Die Werte bewegen sich dabei im Bereich von $0,5km \leq r \leq 3,0km$ in $0,5km$ Schritten. Untersucht wurde die mittlere Abweichung der Anzahl der normalen Stationen über den Tagesverlauf in Bezug auf das Referenzszenario ohne selbstorganisierendes Verhalten. Dabei wird deutlich, dass bei einem Kommunikationsradius von $1,5$ km eine maximale Abweichung (in Form einer Verbesserung) von $10,7\%$ erreicht wird (siehe Abbildung 7.9). Wird eine Gesamtanzahl von $345$ Stationen betrachtet, so bedeutet das, dass ungefähr $35$ Stationen mehr im normalen Zustand sind. Im Referenzszenario ohne selbstorganisierende Umverteilung der Fahrräder befinden sich mindestens $280$ Stationen im normalen Zustand. Erhöht man diese Anzahl um eine Verbesserung von $10\%$ bezogen auf die Gesamtanzahl an Stationen, ergeben sich daraus $315$ Stationen. Im Umkehrschluss bedeutet dies, dass Anzahl der Stationen die nicht im normalen Zustand sind, um etwa $54\%$ von $65$ auf $30$ reduziert wird.

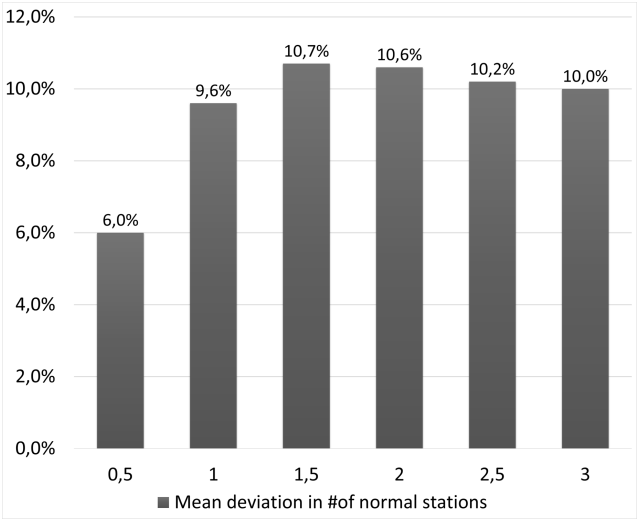



**Abbildung 7.9.:** Mittlere Abweichung der Anzahl der normalen Stationen im Vergleich zu einem Referenzszenario ohne selbstorganisierendes Verhalten - aus (PDR16b, S. 8).

Da der Kommunikationsradius einen direkten Einfluss auf die Entfernung hat, die ein Benutzer potentiell umgeleitet wird, wenn ihm eine alternative Station vorgeschlagen wird, sollte dieser Wert so klein wie möglich gehalten werden, um die Akzeptanz des Ansatzes bei den Benutzern zu erhöhen. Abbildung 7.9 zeigt in diesem Zusammenhang auch, dass ein hoher Kommunikationsradius, unabhängig vom Einfluss auf die Akzeptanz bei den Benutzern, einen negativen Einfluss auf die Qualität der selbstorganisierenden Umverteilung haben kann. Ab einem Kommunikationsradius von $1,5$ km (Höchstwert bei $10,7\%$) verschlechtern sich die Werte für die mittlere Abweichung wieder. Begründet werden kann dieses damit, dass hierbei die Wege der Benutzer vergrößert werden. Dadurch erhöht sich auch die Dauer der

Ausleihen, also auch die Zeit, in der die Fahrräder verwendet werden und nicht für andere Ausleihen zur Verfügung stehen. Zusätzlich verringert sich bei einem höheren Kommunikationsradius die Dichte der Fahrräder in dem Bereich, in dem sie tatsächlich benötigt werden. Zum Beispiel kann es bei einem zu hohen Kommunikationsradius dazu kommen, dass ein Benutzer aus dem Stadtzentrum in die Vororte umgeleitet wird und das Fahrrad somit nicht mehr verfügbar ist, um die hohe Nachfrage innerhalb des Stadtzentrums zu bedienen.

### Strukturelle Adaption

In der vorangegangenen Betrachtung des selbstorganisierenden Verhaltens wurden zwei Dinge deutlich: Zunächst, dass der Erfolg der selbstorganisierenden Umverteilungsmaßnahmen maßgeblich auf der Kooperationsbereitschaft der Benutzer beruht. Wenn die Benutzer nicht bereit sind, einen Umweg in Kauf zu nehmen, wenn ihnen eine alternative Ausleih- oder Rückgabestation vorgeschlagen wird, funktioniert der Ansatz nicht. Dabei ist anzunehmen, dass bei aller Kooperationsbereitschaft die grundsätzliche Akzeptanz des Systems bei den Benutzern leidet, wenn diese zu oft mit einer Umleitung konfrontiert werden. Sinnvoller erscheint, dass System entsprechend zu konfigurieren, sodass die selbstorganisierende Umverteilung der Fahrräder durch die Benutzer nur angewendet wird, um kritische Auslastungssituationen zu lösen. Des Weiteren wurde im Vorangegangenen deutlich, dass es im Bezug auf den Kommunikationsradius als Koordinationsparameter entscheidend wie herausfordernd ist, einen geeigneten Wert zu finden. Ein zu niedriger ebenso wie ein zu hoher Wert können sich dabei negativ auf die mögliche zu erreichende Verbesserung auswirken.

In beiden Fälle würde das selbstorganisierende Verhalten von einer Adaption der zugrundeliegenden Koordinationsprozesse, wie sie in den Abschnitten 4.4 und 6.3 vorgeschlagen wurde, profitieren. Betrachtet man den Fall der Wahl eines geeigneten Kommunikationsradius als Koordinationsparameter, so gibt es zwei Möglichkeiten, um diese Herausforderung zu lösen. Zunächst kann im Rahmen eines iterativen Entwicklungsprozesses, unterstützt von Simulationen, ein Parameter-Tuning stattfinden, bei dem unterschiedliche Werte ausprobiert werden, bis ein optimaler Wert für den Koordinationsparameter gefunden wurde. Dieses Vorgehen entspricht dem *SodekoVS* Vorgehensmodell [SBP+09] (vgl. Abschnitt 3.2.3) und wurde im Grunde im Vorangegangenen angewandt, als mehrere Simulationen mit unterschiedlichen Kommunikationsradien durchgeführt wurden, um den Einfluss dieses Parameters zu untersuchen und dabei einen optimalen Wert zu finden (vgl. Abbildung 7.9). Eine weitere Möglichkeit wäre eine *parametrische* Adaption, bei der automatisiert zur Laufzeit ein optimaler Koordinationsparameter aus einer Reihe von vorkonfigurierten, antizipierten Alternativen ausgewählt wird. Entscheidender als ein optimaler Koordinationsparameter scheint für die grundlegende Akzeptanz des selbstorganisierenden Umverteilungsansatzes bei den

Benutzern allerdings zu sein, diese nicht permanent mit Umleitungen zu alternativen Stationen zu konfrontieren, sondern dieses Verhalten nur zu benutzen, um besonders kritische Situationen zu lösen. Dieser Gedanke wurde in [PDR16c] umgesetzt: Zunächst findet dabei eine klassische Umverteilung der Fahrräder durch LKWs statt und erst, wenn durch diese Umverteilungsmaßnahme keine ausreichend guten Ergebnisse mehr erzielt werden können, wird zur selbstorganisierenden Umverteilung der Fahrräder durch die Benutzer umgeschaltet.

Im Rahmen von [PDR16c] wurde das Bike-Sharing Beispiel daher zunächst um zwei neue Agententypen erweitert: Der LKW-Operator ist ein immobiler Agent, der eine Reihe von LKW-Agenten losschickt, um Fahrräder von vollen Stationen dahin zu transportieren, wo sie benötigt werden. Dazu senden die Fahrradstationen, ähnlich wie auch beim dezentralen Koordinationsansatz zur selbstorganisierenden Umverteilung durch die Benutzer, periodisch ihre aktuelle Auslastungsrate an den LKW-Operator. Dieser verwendet diese Informationen, um daraufhin Aufträge für die LKWs zu erzeugen. Ein solcher Auftrag instruiert einen LKW darüber, wie viele Fahrräder er von welcher Station zu welcher anderen Station transportieren soll. Der Operator verwendet dabei einen einfachen Aufgabenplaner, der die Stationen mit den höchsten Auslastungsraten mit denen mit den niedrigsten Auslastungsraten verbindet. Daraus resultiert eine Prioritätenliste an Aufträgen, die nacheinander von den LKWs abgearbeitet werden. Dazu wurde ein weiterer Koordinationsprozess realisiert, der in Quelltext 7.4 dargestellt ist. Dabei wird die bereits bekannte `bikestation` Rolle um die `truck_operator` Rolle für den LKW-Operator Agenten ergänzt. Die Aussendung der Auslastungsraten als Koordinationsinformationen auf Seite der Fahrradstationen verläuft dabei analog zu dem zuvor behandelten Koordinationsprozess. Auf Seite des `truck_operator` wird der Empfang einer solcher Koordinationsinformation auf das `TruckOperatorStatusUpdateEvent` Koordinationsereignis abgebildet. Der Erhalt eines solchen Koordinationsereignisses wird innerhalb der `handleCoordinationEvent(CoordinationEvent<?> ce)` Methode des `ICoordinatable` Interfaces, das vom `TruckOperator` implementiert wird, verarbeitet. Zur Ausbreitung der Koordinationsinformationen wird von diesem Koordinationsprozess das `TruckOperatorCoordinationMedium` verwendet. Da es nur einen LKW-Operator in diesem Beispiel gibt, enthält dieses Koordinationsmedium eine direkte Referenz auf den Koordinationsendpunkt des LKW-Operators und kann diesen so direkt adressieren.

Die Auswirkungen der bereits bekannten selbstorganisierenden Umverteilung von Fahrrädern und der LKW-basierten Umverteilungsmaßnahmen sind in Abbildung 7.10 gegenübergestellt. Dabei wird wieder die Anzahl der Stationen im *normalen* (weder voll noch leer) Zustand über den Verlauf des Tages dargestellt. Der Kommunikationsradius für die selbstorganisierende Umverteilung betrug in diesem Fall 1 km und dem LKW-Operator standen 10 LKWs zur Umverteilung der Fahrräder zur Verfügung. Betrachtet man die Abbildung, so wird deutlich, dass die Anzahl der normalen Statio-

```
<coordination-description context="Bikesharing">
 <role-definitions>
  <role name="bikestation">
   <components>
    <component class="de.haw.c4das.bikesharing.simulation.Bikestation" />
   </components>
  </role>
  <role name="truck_operator">
   <components>
    <component class="de.haw.c4das.bikesharing.simulation.truck.
         TruckOperator" />
   </components>
  </role>
 </role-definitions>
 <deco-link-definitions>
  <!-- Bikestation status update link -->
  <deco-link id="bs-update-truck" decoupled="false">
   <from role="bikestation" event="de.haw.c4das.bikesharing.simulation.
        coordination.commRange.BikeStationStatusUpdateEvent" />
   <medium class="de.haw.c4das.bikesharing.simulation.coordination.truck.
        TruckOperatorCoordinationMedium" self="false" active="true" />
   <to role="truck_operator" event="de.haw.c4das.bikesharing.simulation.
        coordination.truck.TruckOperatorStatusUpdateEvent" />
  </deco-link>
 </deco-link-definitions>
</coordination-description>
```

**Quelltext 7.4:** XML-basierte Beschreibung des Koordinationsprozesses zur Information des LKW-Operators über die aktuellen Auslastungsraten durch die Fahrradstationen im Bike-Sharing Beispiel.

nen über den Verlauf des Tages höher ist, wenn die selbstorganisierende Umverteilungsstrategie angewendet wird. Wie bereits erwähnt, setzt diese Strategie aber eine hohe Kooperationsbereitschaft auf Seiten der Benutzer voraus, einer vorgeschlagenen Umleitung zu einer alternativen Station auch zu folgen. Daher ist im Bezug auf die Akzeptanz der Umverteilungsstrategien bei den Benutzern die LKW-basierte Strategie vorzuziehen.

Im Rahmen einer Anwendungsstudie [PDR16c] wurde das Konzept der strukturellen Adaption von Koordinationsprozessen auf diesen Fall angewendet. Die Idee dahinter ist die, dass das System zunächst mit dem LKW-basierten Koordinationsprozess ausgeführt wird. Wenn im Fall eines besonders hohen Andrangs ein Performanzverlust in Form von nicht optimalen Auslastungsraten beobachtet wird, adaptiert das System die Struktur der Koordinationsprozesse und wechselt von der LKW-basierten zu der selbstorganisierenden Umverteilungsstrategie. Um eine solche strukturelle Adaption umzusetzen, wurde zunächst die Auslastungsrate $a$ der Stationen auf ein Adaptionselement abgebildet (vgl. Abschnitt 6.3.1). Sobald sich dieser Wert in einer der Stationen ändert, wird der zugehörige *Adaptation Handler* über diese Wertveränderung informiert (vgl. Systemarchitektur in Abbildung 6.13). Der *Adaptation Handler* überprüft entsprechend, ob durch die Wertveränderung der strukturelle Adaptionsprozess ausgelöst werden soll. Der Normalzustand ist dabei definiert als

$$\forall a(0,25 \leq a \leq 0,75). \tag{7.1}$$

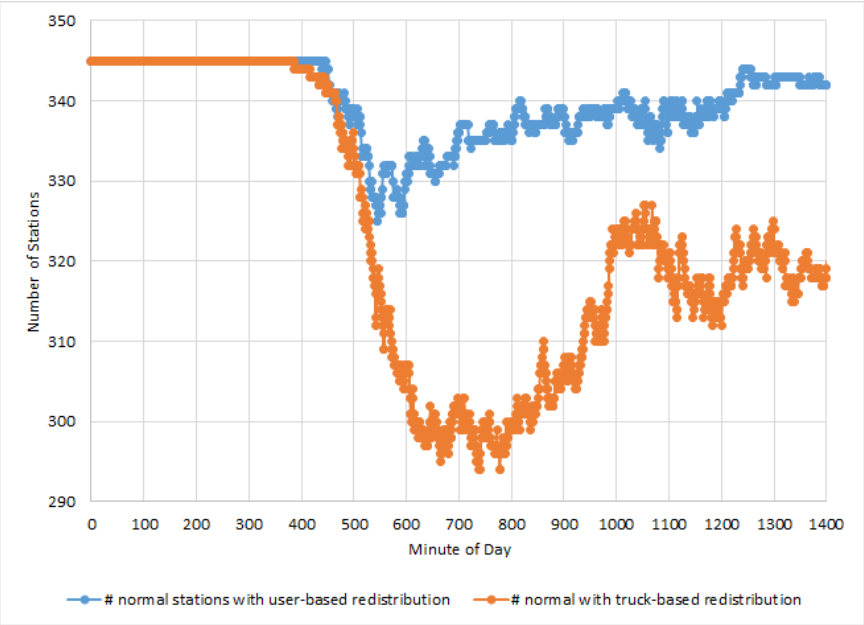



**Abbildung 7.10.:** Vergleich der Auswirkung der selbstorganisierenden und LKW-basierten Umverteilungsmaßnahmen im Bike-Sharing Beispiel - aus (PDR16c, S. 4)

Eine Verletzung des Normalzustandes durch die Adaptationsbedingung $C_A$ löst den Adaptionsprozess aus. $C_A$ ist dabei definiert als

$$C_A = \{a | a \geq 0,75 \vee a \leq 0,25\}. \tag{7.2}$$

Das bedeutet, dass eine Station bei einer Auslastungsrate von $75\%$ oder mehr bzw. $25\%$ oder weniger die strukturelle Adaption der Koordinationsprozesse den anderen Stationen vorschlägt, um somit zur selbstorganisierenden Umverteilung der Fahrräder zu wechseln. Eine Station stimmt einer vorgeschlagenen Adaption zu, wenn die Schwellwertbedingung $C_T$ erfüllt ist. $C_T$ ist dabei definiert als

$$C_T = \{a | a \geq 0,66 \vee a \leq 0,33\}. \tag{7.3}$$

Eine Station stimmt also einer vorgeschlagenen Adaption zu, wenn sie selber zu $66\%$ oder mehr bzw. zu $33\%$ oder weniger ausgelastet ist. Der Adaptionsprozess wurde ferner so konfiguriert, dass die Adaption durchgeführt wird, sobald $50\%$ oder mehr der Stationen einem solchen negativen Trend ausgesetzt sind. Die XML-basierte Beschreibung, die diesen strukturellen Adaptionsprozess instruiert, ist in Quelltext 7.5 dargestellt. Die `bikestation` Rolle referenziert dabei die Rolle der Stations-Agenten, wie sie in Quelltext 7.3 und 7.4 definiert wurde. Über das `occupancy` Element wird angegeben, dass in den Komponenten das entsprechende Adaptionselement vorhanden sein muss, das die Auslastungsrate abbildet. Die `trigger` Kondition beschreibt die Bedingung aus Gleichung 7.2, die den Adaptionsprozess auslöst. Innerhalb des `voting`-Tags wird der verteilte Abstimmungsprozess konfiguriert und u.a. mit der `thresholdCondition` die Schwellwertbedingung definiert, mit deren

Eintreten eine Komponente einer vorgeschlagenen Adaption zustimmt (vgl. Gleichung 7.3). In den Aktionen (`actions`) ist schließlich definiert, dass der Koordinationsprozess für die LKW-basierte Umverteilung im Falle der Adaption deaktiviert und dafür der Koordinationsprozess für die selbstorganisierende Umverteilung aktiviert wird. Über die dort angegebenen `deco-link-id`s werden die entsprechenden Koordinationsprozesse, wie sie in Quelltext 7.3 und 7.4 definiert sind, referenziert.

```
1 <adaptation-definitions>
2  <adaptation id="structural_adaptation" role="bikestation" element="
       occupancy" single="true" decoupled="false">
3   <trigger condition="occupancy &gt; 0.75 || occupancy &lt; 0.25"/>
4   <voting minAnswers="345" timeout="0" quorum="0.50" thresholdCondition="
        occupancy &gt; 0.66 || occupancy &lt; 0.33"/>
5   <actions>
6    <action deco-link-id="bs-update" active="true"/>
7    <action deco-link-id="bs-update-truck" active="false"/>
8   </actions>
9  </adaptation>
10 </adaptation-definitions>
```

**Quelltext 7.5:** XML-basierte Beschreibung der strukturellen Adaption im Bike-Sharing Beispiel.

Das System verhält sich nun wie folgt: Zum Start des Szenarios ist die LKW-basierte Umverteilung der Fahrräder aktiv. In dieser Konfiguration nimmt die Anzahl der normalen Stationen im Verlauf des Tages zunächst ab. Ungefähr um Minute $650$ findet daher die strukturelle Adaption statt. In diesem Moment wechselt das System von der LKW-basierten Umverteilung der Fahrräder zur selbstorganisierenden, indem die entsprechenden Koordinationsprozesse ausgetauscht werden. Nach diesem Wechsel der Koordinationsprozesse steigt die Zahl der normalen Stationen im weiteren Verlauf des Tages wieder an. Das unerwünschte Systemverhalten im Bezug auf die Auslastungsraten der Stationen wurde also durch die strukturelle Adaption der Koordinationsprozesse überwunden. Das entsprechende Verhalten ist in Abbildung 7.11 dargestellt. Die Kurve zeigt dabei die Anzahl der Stationen, die sich im normalen Zustand befinden und der senkrechte Strich stellt den Zeitpunkt der durchgeführten strukturellen Adaption dar.

Abbildung 7.11 zeigt nun, wie sich das System in diesem Fall verhält. Dabei stellt die rote Kurve die Anzahl der Stationen dar, die sich im normalen Zustand befinden. Zum Start des Szenarios ist dabei die LKW-basierte Umverteilung der Fahrräder aktiv. In dieser Konfiguration nimmt die Anzahl der normalen Stationen im Verlauf des Tages zunächst ab. Ungefähr um Minute $650$ findet dann die strukturelle Adaption statt (dargestellt durch den senkrechten Strich). In diesem Moment wechselt das System von der LKW-basierten Umverteilung der Fahrräder zur selbstorganisierenden, indem die entsprechenden Koordinationsprozesse ausgetauscht werden. Nach diesem Wechsel der Koordinationsprozesse steigt die Zahl der normalen Stationen im weiteren Verlauf des Tages wieder an. Das unerwünschte Systemverhal-

ten im Bezug auf die Auslastungsraten der Stationen wurde also durch die strukturelle Adaption der Koordinationsprozesse überwunden. Zur besseren Verständlichkeit der Abbildung und des strukturellen Adaptionsprozesses sollte an dieser Stelle noch darauf hingewiesen werden, dass die normalen Stationen, die in der Abbildung dargestellt sind, solche sind, die weder voll noch leer sind. Das bedeutet also, dass es sich um Stationen handelt, deren Auslastungsrate zwischen $0 < a < 1$ liegt. Daher hängt die Darstellung der normalen Stationen in der Abbildung nicht direkt mit den Stationen, die eine strukturelle Adaption auslösen oder dieser zustimmen würden zusammen. Dementsprechend befinden sich zum Zeitpunkt der durchgeführten strukturellen Adaption gemäß der Abbildung auch nur etwa $15\%$ der Stationen nicht im normalen Zustand.

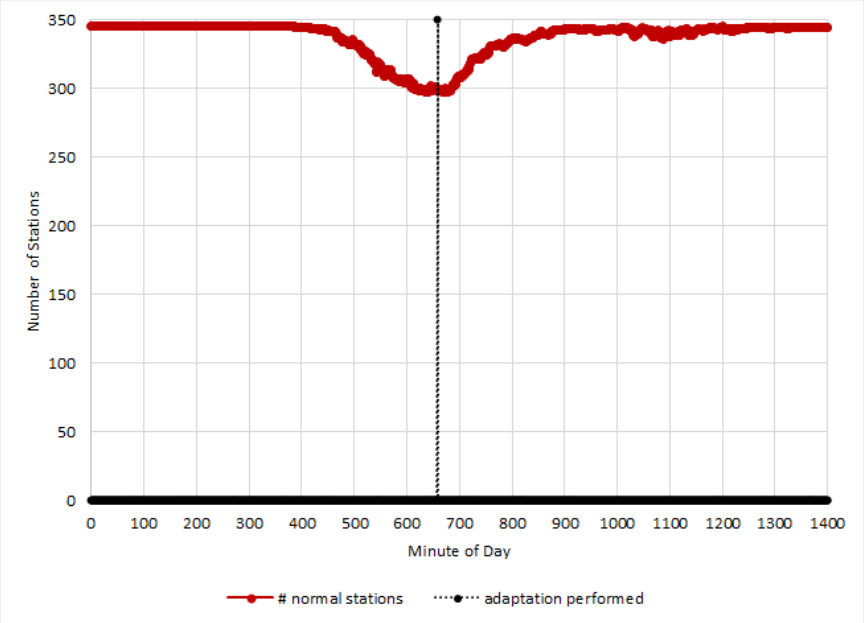



**Abbildung 7.11.:** Auswirkung und Nachweis der Strukturelle Adaption der Koordinationsprozesse im Bike-Sharing Beispiel - nach (PDR16c, S. 4).

### Zusammenfassung und Bewertung

In den Anwendungsstudien, die im Rahmen von [PDR16b] und [PDR16c] durchgeführt wurden, wurde anhand des Beispiels des selbstorganisierenden Fahrradverleihsystems gezeigt, wie das in Abschnitt 6.2 vorgestellte dezentrale Koordinationsframework verwendet werden kann, um selbstorganisierendes Verhalten auf Basis von dezentralen Koordinationsprozessen umzusetzen. Dabei wurde insbesondere darauf eingegangen, wie sich dezentrale Koordinationsprozesse mit Hilfe der deklarativen Beschreibungssprache modellieren lassen und welche technischen Integrationsschritte aus Entwicklersicht durchgeführt werden müssen, um generische Softwarekomponenten mit der entsprechenden Fähigkeit zur Koordination auszustatten. Damit konnte

im Sinne einer Validierung gezeigt werden, dass das entwickelte Koordinationsframework geeignet ist, um dezentrale Koordinationsprozesse zu realisieren und damit selbstorganisierendes Verhalten in einem verteilten System umzusetzen. Weiterhin wurde dabei gezeigt, wie das Konzept der strukturellen Adaption von Koordinationsprozessen mit dem entwickelten Koordinationsframework umgesetzt werden kann, sodass die Struktur der selbstorganisierenden Dynamik im Rahmen einer zur Designzeit antizipierten Adaption automatisiert zur Laufzeit angepasst werden kann, sobald das System nicht mehr das vorgesehene, selbstorganisierende Verhalten aufweist.

## 7.2. Bewertung des eigenen Ansatzes

In diesem Abschnitt erfolgt eine Bewertung des in Abschnitt 6.2 präsentierten dezentralen Koordinationsframeworks. Dazu wird der entwickelte Ansatz zunächst mit anderen, existierenden Ansätzen zur Umsetzung von dezentraler Koordination in verteilten Systemen verglichen, bevor eine Bewertung, besonders im Hinblick auf die in der Einleitung formulierten Forschungsfragen, erfolgt.

### 7.2.1. Vergleich mit existierenden Ansätzen

Im Folgenden geht darum, das in Abschnitt 6.2 vorgestellte Framework zur dezentralen Koordination mit existierenden Ansätzen, die eine ähnliche Zielsetzung verfolgen, zu vergleichen. Grundsätzlich gibt es viele verschiedene Ansätze, die sich mit der Herausforderung, selbstadaptive bzw. selbstorganisierende Systeme zu entwickeln, befassen. Dabei werden im Rahmen von Forschungsrichtungen wie *Autonomic Computing* [KC03] oder *Organic Computing* [BMMSP06] Ansätze bereitgestellt, die diese Herausforderungen systematisch adressieren. Diese beiden Ansätze basieren auf unterschiedlichen Arten von Feedback-Schleifen, die i.d.R. auf zentralisierte Kontrollkomponenten zurückgreifen. Solche Feedback-Schleifen sind laut [BMSG+09] ein entscheidendes Design-Element innerhalb eines verteilten System, um Adaptivität zu realisieren (vgl. Abschnitt 3.1.1). Derartige Feedback-Schleifen bestehen normalerweise aus drei Hauptkomponenten:

1. Sensoren, die das Verhalten und den Zustand der Komponenten bzw. der Umgebung, in der die Komponente situiert ist, beobachten.
2. Aktoren, die die Konfiguration des Systems verändern können, was zu Änderungen an dem Verhalten der Komponenten führt.
3. Einer Berechnungseinheit, die als Konnektor zwischen den Eingaben des Systems (Sensoren) und den Ausgaben (Aktoren) dient. Die Ausprägung dieser Entität kann dabei im Hinblick auf ihre interne Architektur und die Fähigkeiten sehr unterschiedlich sein (vgl. u.a. [MBAG06]).

Ein wichtiger Aspekt, um nichtfunktionale Anforderungen wie Ausfallsicherheit, Robustheit und Skalierbarkeit in großen verteilten Systemen

sicherzustellen, ist die Verwendung von dezentralen Kontroll- bzw Koordinationskonzepten. In [WMA10] wird dabei zwischen dezentralen Lösungen und ihren zentralen Gegenstücken unterschieden und herausgearbeitet, dass erstere aufgrund ihrer dezentralen Natur deutlich besser geeignet sind, um die erwähnten nichtfunktionalen Anforderungen zu erfüllen. Insgesamt lässt sich feststellen, dass die dezentrale Koordination von adaptiven Systeme ein weitverbreitetes Forschungsfeld ist, in dem zahlreiche Anwendungen und Ansätze existieren. Im Folgenden werden einige dieser Ansätze präsentiert und mit dem im Rahmen dieser Dissertation entwickelten dezentralen Koordinationsframework (*DeCoF*) verglichen.

Die Autoren von [NVCO10] präsentieren z.B. einen selbstorganisierenden Koordinationsansatz, bei den die Koordinationsmechanismen von chemischen Reaktionen, wie sie in der Natur vorkommen, inspiriert sind. Dieser *TuCSoN Coordination Space* genannte Ansatz basiert auf einer Vielfalt von unabhängigen Kommunikationsabstraktionen, sogenannten *Tuple Center*. Diese *Tuple Center* können dabei auf verschiedenen Netzwerkknoten verteilt sein und von Software-Agenten verwendet werden, um miteinander zu interagieren. Die *Tuple Center* übernehmen bei *TuCSoN* dabei die Aufgabe der Koordinationsmedien und reichern den Koordinationsraum des Systems mit spezifischen Verhalten an. Die *Tuple Center* sind also Kommunikationsabstraktionen, deren Verhalten die Koordinationsfunktionalität des Systems abbildet. Die *Tuple Center* werden in diesem Ansatz so ähnlich verwendet wie die Koordinationsmedien im *DeCoF*. Der *TuCSoN* Ansatz propagiert dabei ebenfalls eine saubere Trennung zwischen Aspekten, die die Koordination und Aspekten, die die eigentliche Anwendungslogik betreffen, nach [GC92]. Koordination wird in diesem Ansatz also ebenfalls als orthogonaler Aspekt im Bezug auf die eigentliche Berechnungslogik bei der Realisierung von verteilten Systemen gesehen (vgl. Abschnitt 3.1.2).

Ein weiterer, ähnlicher Ansatz zur dezentralen Koordination, der wie *TuCSoN* auf einem Tuple Space Ansatz beruht, ist *DTuples* [JXJY06]. *DTuples* benutzt dabei eine Peer-to-Peer basierte Middleware, die auf einer verteilten Hashtabelle (engl. *distributed hash table*, DHT) [GG09] basiert. Das Ziel dieses Ansatzes ist es, die Vorteile von Peer-to-Peer Systemen im Hinblick auf Eigenschaften wie Robustheit und Skalierbarkeit mit dem *Tuple Space* Konzept aus *Linda* [GB82] zu kombinieren. Da gängige *Tuple Space* Modelle wie *Linda* einen zentralen Raum zur Kommunikation und Koordination verwenden, leiden sie unter Skalierungsproblemen und der kritischen Abschnittsproblematik (engl. *single point of failure*). Peer-to-Peer Systeme überwinden diese Probleme durch ihren dezentralen Ansatz. *DTuples* verbindet daher die Vorteile von Peer-to-Peer Systemen mit der Bereitstellung von Sprachelementen (Primitiven) zur Koordination aus dem *Tuple Space* Ansatz.

Einen unterschiedlichen Weg verfolgt der *Agents and Artifacts* (A&A) Meta-Modell Ansatz [RVO08], der sich speziell auf die *Jason* [BHW07] Agentenplattform konzentriert. Gemäß diesem Ansatz besteht ein Multiagentensystemen aus Agenten und sogenannten Artefakten. Artefakte sind Elemen-

te der Umgebung, die von den Agenten verwendet werden können. Um ein solches Multiagentensystem mit der Fähigkeit zur Koordination auszustatten, können spezifische Koordinationsartefakte abgeleitet werden. Diese Artefakte können z.B. die *Blackboard*-Kommunikationsmetapher [Nii86], wie sie auch im *Tuple Space* Konzept Anwendung findet, aufgreifen. Das A&A Modell bedingt dabei allerdings, dass die Agenten diese Koordinationsartefakte explizit verwenden. Dies impliziert, dass die Koordination nicht transparent für die Agenten zur Verfügung steht. Koordination muss somit ein expliziter Teil der Anwendungslogik sein. Der Ansatz des A&A Meta-Modells wurde im Rahmen von [RPV11] erweitert. Hier wird ein Konzept zur Programmierung der Umwelt eines Multiagentensystems vorgestellt. Das daraus resultierende *CArtAgO* Framework stellt dabei ein konkretes Programmiermodell zur Verfügung, um Multiagentensysteme auf Basis des A&A Modells mit Koordinationsfähigkeiten auszustatten.

Ein formaler Ansatz zur Programmierung von autonomen Systemen wurde mit der Einführung der *Software Component Ensemble Language* (SCEL) in [NLPT14] beschrieben. Dabei werden eine Reihe von Programmierabstraktionen vorgeschlagen, mit denen Verhalten, Wissen und entsprechende Aggregationen davon repräsentiert werden können. Mit Hilfe dieser Programmierabstraktionen soll die Entwicklung von kontext- und selbst-wahrnehmenden Anwendungen mit der Fähigkeit zur Adaption unterstützt werden. Neben der Menge von linguistischen Programmierabstraktionen wird dazu eine Java Implementation bereitgestellt. Im Gegensatz zu dem dezentralen Koordinationsframework, das im Rahmen dieser Dissertation entwickelt wurde, erfolgt die Beschreibung des adaptiven Verhaltens der Anwendung dabei in imperativer anstelle deklarativer Weise. Außerdem unterstützt der Ansatz keine explizite Trennung zwischen Koordinations- und Anwendungslogik nach [GC92].

Der Vollständigkeit halber wird an dieser Stelle auch kurz auf den *SodekoVS* Ansatz [SBP+09] eingegangen, der in bereits in Abschnitt 3.2.3 ausführlicher beschrieben wurde. Das im Rahmen dieser Arbeit entwickelte dezentrale Koordinationsframework basiert auf diesem Ansatz, der mit dem Ziel entwickelt wurde, die software-technische Realisierung von selbstorganisierenden Systemen durch die Bereitstellung von Koordinationsmechanismen als wiederverwendbare Designelemente zu unterstützen. Der *SodekoVS* Ansatz bietet dabei ein Programmiermodell zur systematischen Modellierung von selbstorganisierendem Verhalten, das durch eine deklarative Beschreibungssprache für Koordinationsprozesse (*MASDynamics* [SR09b]) und eine Referenzarchitektur für die *Jadex* Agentenplattform [PBJ13] (*DeCoMAS* [SR09a]) unterstützt wird. Bei dem *Coordination Space* Ansatz [VSL+10] handelt es sich um eine Weiterentwicklung der *DeCoMAS* Referenzarchitektur, bei der die Koordination in einen speziellen Teil der Umgebung des zugrundeliegenden Multiagentensystems ausgelagert wird. Dieser Ansatz verwendet daher ebenfalls die *MASDynamics* Sprache zur deklarativen Beschreibung der Koordinationsprozesse und wurde im Rahmen von [PVR13] um ein Verteilungskonzept er-

weitert, mit dem Ziel, vorherige Limitierungen im Bezug auf die Verteilbarkeit zu überwinden (vgl. Abschnitt 4.3).

Tabelle 7.2 zeigt nun einen Vergleich der in diesem Abschnitt vorgestellten existierenden Ansätze. Ergänzt ist die Tabelle dabei um den *Coordination Space* Ansatz, wie er in Abschnitt 4.3 präsentiert wurde, sowie den Ansatz zur dezentralen Koordination von *Aktiven Komponenten* aus Abschnitt 6.1 („*Decentralized Coordination of Jadex Active Components*“, DeCoJAC). Die Gegenüberstellung in Tabelle 7.2 vergleicht die unterschiedlichen Ansätze dabei nach fünf unterschiedlichen Kriterien:

1. *Trennung zwischen Koordinations- und Anwendungslogik*, hier werden die Ansätze darin unterschieden, ob sie eine explizite Trennung zwischen der Koordinations- und Anwendungsfunktionalität, wie sie von [GC92] gefordert wurde, unterstützen.

2. *Agenten- oder Komponentenunterstützung*, bei diesem Kriterium werden die Ansätze nach der Systemarchitektur, die sie unterstützen, unterschieden. Die Unterscheidung erfolgt dabei zunächst danach, ob der Ansatz speziell zur Koordination von Agenten entwickelt wurde oder generische Software-Komponenten unterstützt werden. Falls ersteres der Fall ist, wird noch nach der unterstützten Agentenplattform und dem Typ der Agenten unterschieden.

3. Die *Art der Koordinationsbeschreibung* unterscheidet die Ansätze danach, in welcher Form die Koordinationsprozesse beschrieben werden. Hierbei wird zwischen einer imperativen und deklarativen Form der Beschreibung unterschieden. Bei einer imperativen Form wird von dem Entwickler explizit der Algorithmus des Koordinationsprozesses beschrieben, es wird also der Lösungsweg vorgegeben und angegeben, welche einzelnen Schritte nacheinander ablaufen sollen. Bei der deklarativen Beschreibungsform von Koordinationsprozessen steht die Beschreibung dessen im Vordergrund. Hier konzentriert sich die Beschreibung daher eher darauf *Was* koordiniert werden soll und weniger auf das *Wie* (imperative Form).

4. Das *Koordinationsmodell* vergleicht Ansätze nach dem Modell, das dem Koordinationsansatz zugrunde liegt.

5. Schließlich werden die verschiedenen Ansätze noch dahingehend unterschieden, ob sie eine strukturelle Adaption der Koordinationsprozesse, wie sie in Abschnitt 4.4 vorgestellt wurde, unterstützen.

Die Tabelle zeigt dabei insbesondere, wie der *DeCoF* Ansatz als konsequente Weiterentwicklung der Vorarbeiten das ursprüngliche Konzept zur Realisierung von selbstorganisierenden Verhalten in verteilten Systemen auf Basis von dezentraler Koordination ergänzt. Dieses geschieht durch eine Loslösung vom Paradigma der Multiagentensysteme, durch die es möglich

**Tabelle 7.2.:** Vergleich unterschiedlicher Ansätze zur Realisierung von dezentraler Koordination in verteilten Systemen.

| Ansatz/ Framework | Trennung zwischen Koordinations- und Anwendungs-logik | Agenten- oder Komponenten-unterstützung | Art der Koordinations-beschreibung | Koordinationsmodell | Unterstützung strukturelle Adaptionen |
|---|---|---|---|---|---|
| TuCSoN | Ja | Allgemeine Software-Komponenten | Imperativ | Dedizierter Tuple Space zur Koordination | Nein |
| DTuples | Ja | Allgemeine Software-Komponenten | Imperativ | Dedizierter Peer-to-Peer Tuple Space zur Koordination | Nein |
| CArtAgO (A&A) | Nein | Jason BDI-Agenten | Imperativ | Bereitstellung bestimmter Koordinationsartefakte | Nein |
| SCEL | Nein | Allgemeine Software-Komponenten | Imperativ | Linguistische Programmierabstraktionen | Nein |
| DeCoMAS | Ja | Jadex BDI-Agenten | Deklarativ | Koordinationsprozesse auf Basis von Koordinations-endpunkten und Koordinations-medien | Nein |
| Coordination Space (erweitert durch [PVR13]) | Ja | Jadex BDI- und Mikro-Agenten | Deklarativ | Koordinationsprozesse auf Basis von Koordinations-endpunkten und Koordinations-medien | Ja (durch [PRD15]) |
| DeCoJAC | Ja | Jadex Aktive Komponenten | Deklarativ | Koordinationsprozesse auf Basis von Koordinations-diensten und Koordinationsdienst-selektoren | Nein |
| DeCoF | Ja | Allgemeine Software-Komponenten | Deklarativ | Koordinationsprozesse auf Basis von Koordinations-endpunkten und Koordinations-medien | Ja |

wird, auch komplexe, verteilte Systemen, die aus unterschiedlichen heterogenen Softwarekomponenten bestehen, wie sie z.B. im Umfeld der Co-Simulation eine Rolle spielen (vgl. Kapitel 5), mit der Fähigkeit zur Koordination auszustatten. Weiterhin zeigt die Tabelle den grundlegenden Unterschied zwischen dem *DeCoF* Ansatz (und seiner Vorarbeiten) und den anderen Ansätzen, der in Art der Koordinationsbeschreibung liegt. Während die anderen Ansätze auf eine imperative Beschreibung der Koordinationslogik setzen, erfolgt die Beschreibung der Koordinationslogik in *DeCoF* (und seinen Vorarbeiten) in deklarativer Form. Hier steht bei der Beschreibung der Koordinationsprozesse das *Was* im Vordergrund und nicht das *Wie*.

### 7.2.2. Bewertung im Hinblick auf die Forschungsfragen

Im Folgenden werden zur Bewertung der im Rahmen dieser Dissertation erzielten Ergebnisse zunächst die Forschungsfragen, die bereits eingangs in der Einleitung (vgl. Abschnitt 1.2) formuliert wurden, wieder aufgegriffen:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen, um koordinierbar zu sein?

2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

3. Wie kann in einem selbstorganisierenden System dynamischen Ineffizienzen in Form eines unerwünschten Verhaltens entgegen gewirkt werden?

Die ersten beiden Forschungsfragen wurden bereits im Rahmen von Abschnitt 6.2.5 beantwortet. Der Vollständigkeit halber werden die dort beschrieben Ergebnisse an dieser Stelle erneut aufgegriffen. Bezug nehmend auf die erste Frage lässt sich feststellen, dass es einen grundsätzlichen Unterschied zwischen der Koordination von Komponenten gibt, die im Rahmen einer Ausführungsumgebung *verwaltet* werden und solchen, die *nicht verwaltet*, also losgelöst von einer speziellen Ausführungsumgebung ausgeführt werden. Ein Beispiel für solche *Managed*-Komponenten sind Software-Agenten, die im Rahmen einer Agentenplattform ausgeführt und verwaltet werden. Solche Komponenten bzw. die Ausführungsumgebungen, für die sie entwickelt werden, bieten dabei spezielle Modellierungselemente an, mit denen zur Entwicklungszeit das Verhalten der Komponenten bestimmt werden kann und über deren Observierung zur Ausführungszeit der Zustand der Komponenten bestimmt werden kann. Dadurch können Zustände oder Ereignisse, die für die Koordination relevant sind, observiert und benutzt werden, um Koordinationsprozesse auszulösen. Bei nicht verwalteten Komponenten (*unmanaged*) fehlt es an solchen implizit vorhandenen Modellierungselementen. In Abschnitt 6.2 wurde daher das Konzept expliziter Koordinationsereignisse präsentiert. Das sind Ereignisse, die von den entsprechenden Komponenten ausgelöst werden,

sobald ein für die Koordination relevanter Zustand erreicht wird. Dadurch wird allerdings die strikte Trennung zwischen Koordinations- und Anwendungslogik, wie sie von [GC92] gefordert wird, etwas aufgeweicht. Allerdings wird durch das Konzept des Koordinationsbus, das ebenfalls in Abschnitt 6.2 beschrieben wurde, erreicht, dass die Komponenten zwar explizit Koordinationsereignisse auslösen müssen, sie allerdings kein explizites Wissen darüber benötigen, ob sie zum Zwecke der Koordination überwacht und beeinflusst werden. Der Koordinationsbus ermöglicht an dieser Stelle eine lose Kopplung zwischen den zu koordinierenden Komponenten und sogenannten Endpunkten, welche die Komponenten zum Zwecke der Koordination überwachen und beeinflussen (vgl. Abschnitt 6.2). Die Anwendungslogik der Komponenten kann somit unabhängig von der Koordinationslogik realisiert werden und muss nur an geeigneten Stellen Koordinationsereignisse auslösen, sich dann aber nicht weiter um deren Verarbeitung kümmern.

Die zweite Forschungsfrage wurde ebenfalls in Abschnitt 6.2 in Form der dort präsentierten Architektur des dezentralen Koordinationsframeworks beantwortet. Auf konzeptioneller Ebene wurde dabei gezeigt, dass die Koordination von generischen Softwarekomponenten auf Basis der Konzepte der Koordinationsereignisse, des Koordinationsbus sowie der Koordinationsendpunkte und Koordinationsmedien erfolgen kann. Dabei wurde gezeigt, wie die zur Koordination notwendige Interaktionen dieser Komponenten in Form von deklarativ beschriebenen Koordinationsprozessen instruiert werden kann. Auf technischer Ebene erfolgt die Integration dabei über das in dem Abschnitt präsentierte `ICoordinatable` Interface. Diese Schnittstelle muss von Komponenten implementiert werden, wenn sie mit der Fähigkeit zur Koordination ausgestattet werden sollen. Innerhalb der Schnittstelle erfolgt dabei die Komponenten-spezifische Verarbeitung von Koordinationsereignissen, die eine Komponente im Rahmen eines Koordinationsprozesses empfängt. Die Verarbeitung dieser Ereignisse muss dabei berücksichtigen, ob es sich um eine verwaltete Komponente handelt, bei der die Verarbeitung der eingehenden Koordinationsereignisse im Rahmen plattformspezifischer Scheduling-Mechanismen erfolgen muss, oder ob es sich um eine nicht verwaltete Komponente handelt, bei der die Verarbeitung entsprechend dem Komponenteneigenen Scheduling erfolgen muss.

Die dritte Forschungsfrage wurde schließlich in den Abschnitten 4.4 und 6.3 durch das vorgeschlagene Konzept der strukturellen Adaption von Koordinationsprozessen beantwortet. Grundsätzlich hat sich das Konzept der Selbstorganisation seit mehr als einem Jahrzehnt als dezentraler Ansatz zur Entwicklung von selbstadaptiven Systemen etabliert. Allerdings kann dabei beobachtet werden, dass solche selbstorganisierende Systeme unter bestimmten Bedingungen ein unerwünschtes Verhalten aufzeigen können. Um nun solche dynamischen Ineffizienzen zu überwinden, wurde im Rahmen dieser Dissertation der dynamische Austausch bzw. die Rekonfiguration der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden, vorgeschlagen. Ein solcher Austausch der Koordinationsprozesse wird im Kontext dieser

Arbeit als *strukturelle Adaption* bezeichnet, da dadurch die Struktur des selbstorganisierenden Verhaltens selbst angepasst wird. Bei einer Rekonfiguration der Koordinationsprozesse durch eine Anpassungen der Koordinationsparameter wird daher auch von einer *parametrischen Adaption* gesprochen.

In Abschnitt 4.4 wurde dabei eine Lösung zur strukturellen Adaption von Koordinationsprozessen in selbstorganisierenden Multiagentensystemen beschrieben, während Abschnitt 6.3 das Konzept auf die Adaption von Koordinationsprozessen, die mit *DeCoF* (siehe Abschnitt 6.2) realisiert wurden, adaptiert. Abstrahiert man von den jeweiligen spezifischen Eigenschaften der Lösungen, die in den jeweiligen Systemarchitekturen der unterstützten Systeme begründet sind, resultiert das in folgenden Elementen, die für eine solche strukturelle Adaption benötigt werden: Zunächst bedarf es Performanzindikatoren, mit denen festgestellt werden kann, wann eine dynamische Ineffizienz besteht, also wann das System nicht mehr das vorgesehene Verhalten aufweist. In diesem Zusammenhang spricht der Ansatz daher auch von selbstorganisatorischen Performanzindikatoren. Diese weisen keine selbstorganisierenden Eigenschaften auf, sondern geben Ausdruck über die Qualität des selbstorganisierenden Verhaltens. Da der Ansatz der Realisierung von selbstorganisierenden Verhalten auf Basis von Koordinationsprozessen dezentraler Natur ist, ergibt sich dadurch ein Mangel globalen Wissens. Daher handelt es sich bei diesen Performanzindikatoren um lokale Werte, die jeweils abbilden, wie gut die selbstorganisierende Dynamik im System für die jeweilige Komponente funktioniert. Da die selbstorganisierende Dynamik im System unterschiedlichen Einfluss auf die Entitäten in einem verteilten System haben kann, müssen diese einen Konsens darüber finden, ob eine Ineffizienz vorliegt und ob dementsprechend eine strukturelle oder parametrische Adaption durchgeführt werden soll.

Im Rahmen dieser Arbeit wurde dabei ein verteiltes Abstimmungsverfahren vorgeschlagen, mit denen die Entitäten über mögliche Adaptionen abstimmen. Adaptionen sind dabei antizipierter Natur, sie werden also, ähnlich wie die Koordinationsprozesse selber auch, während der Designphase des Systems beschrieben. Dabei werden von dem Entwickler die Bedingungen angegeben, welche die bereits erwähnten Performanzindikatoren referenzieren und unter denen eine Anpassung der Koordinationsprozesse erfolgen soll. Diese Bedingungen werden später im Rahmen des verteilten Abstimmungsverfahrens herangezogen, um zu bestimmen, welche Entitäten zur Laufzeit einer vorgeschlagenen Adaption zu stimmen. Neben den Bedingungen wird ebenfalls beschrieben, wie die Adaption durchgeführt werden soll, also welche Koordinationsprozesse in welcher Form betroffen sind. Die antizipierten Adaptionen passen dabei in den iterativen Entwicklungsprozess, wie er im Rahmen des *SodekoVS* Ansatzes [SBP+09] entwickelt wurde (vgl. auch Abschnitt 3.2.3). Mögliche Adaptionen an den Koordinationsprozessen werden also von den Entwicklern zur Designzeit antizipiert, wobei die Verfeinerung der Bedingungen im Rahmen einer simulationsgetriebenen, iterativen Evaluation der selbstorganisierenden Dynamik und der strukturellen Adaptionen stattfindet.

Im Rahmen dieses iterativen Vorgehens wird also erkannt, unter welchen Bedingungen die selbstorganisierende Dynamik Ineffizienzen aufweist und dafür ein struktureller Adaptionsprozess definiert, der dieses Problem überwindet.

Natürlich bietet die Lösung in ihrem aktuellen Stadium noch Raum für Verbesserungen. So nehmen die antizipierenden Adaptionen dem Problem zwar etwas von seiner Komplexität, andererseits sind sie eingeschränkt in dem, was sie erreichen können. Dynamische Ineffizienzen können dabei nur insofern überwunden werden, wie sie im Vorfeld identifiziert wurden. Visionärer wäre eine Lösung, bei denen das System selbstständig dazu in der Lage wäre, einen geeigneten Weg zu finden, um diese dynamischen Ineffizienzen zu überwinden, z.B. in denen aus einer Menge von möglichen Handlungsalternativen diejenige Kombination gefunden wird, die in der Lage ist, das Problem optimal zu lösen. An dieser Stelle lässt der Ansatz daher noch Raum für weitere Forschungsvorhaben (siehe Abschnitt 8.2.2).

## 7.3. Zusammenfassung

In diesem Kapitel wurden zunächst die beiden Anwendungsbeispiele *MarsWorld Mining Operation* und *Bike-Sharing Washington, D.C.* präsentiert. Im Rahmen der *MarsWorld* Anwendungsstudie wurde das Verhalten einer Gruppe von Robotern koordiniert, die auf dem Mars Erz abbauen. Dabei verwendeten die Roboter, die im Rahmen einer Simulation durch Software-Agenten abgebildet wurden, Koordinationsprozesse, um den Einsatz ihrer unterschiedlichen Fähigkeiten zu koordinieren, sodass sie gemeinsam potentielle Erzvorkommen aufspüren und analysieren konnten, um diese anschließend abzubauen und das geförderte Erz abzutransportieren. Diese Koordinationsprozesse wurden in zwei unterschiedlichen Ausprägungen realisiert. Die erste Koordinationsprozessausprägung basierte auf der Auswahl des nächstgelegenen Agenten vom benötigten Typ und resultierte in der selbstorganisierenden Formation von lokalen Abbaugruppen. Die zweite Ausprägung bestand aus einer einfachen zufallsgesteuerten Auswahl der entsprechenden Agenten und wies somit keine selbstorganisierende Formation von lokalen Abbaugruppen auf. Im Rahmen der Simulation wurde ein Fehler in der Umgebung simuliert, wodurch es den Koordinationsprozessen nicht mehr möglich war, die Positionen der Agenten zu bestimmen. Dadurch kam es zu einer dynamisch auftretenden Ineffizienz der selbstorganisierenden Dynamik. Im Rahmen dieser Anwendungsstudie wurde nun gezeigt, wie die strukturelle Adaption von Koordinationsprozessen verwendet werden kann, um dieses Problem durch den Austausch der Koordinationsprozesse zu lösen. Dabei wurde die Struktur adaptiert, indem die defekten Koordinationsprozessausprägungen durch diejenigen ersetzt wurden, die auf der zufallsgesteuerten Auswahl basieren. Anhand dieser Anwendungsstudie konnte somit im Sinne einer Validierung gezeigt werden, dass die strukturelle Adaption von Koordinationsprozessen geeignet ist, um dynamische Ineffizienzen in selbstorganisierenden Systemen zu überwinden.

Im Rahmen der zweiten Anwendungsstudie wurde anhand der Simulation eines selbstorganisierenden Fahrradverleihsystems gezeigt, wie das dezentrale Koordinationsframework, das in Abschnitt 6.2 präsentiert wurde, verwendet werden kann, um selbstorganisierendes Verhalten auf Basis von dezentralen Koordinationsprozessen zu realisieren. Die Simulation wurde dabei auf Grundlage von Historie-Daten über die Ausleihvorgänge in Washington, D.C. für das Jahr 2014 realisiert. Der Integrationsansatz des entwickelten Frameworks wurde anhand des agenten-basierten Logistiksimulators *RinSim* demonstriert. Das selbstorganisierende Verhalten basierte dabei auf dem Ansatz, die Benutzer des Systems durch geeignete Anreize dazu zu ermutigen, bei der Ausleihe und der Rückgabe von Fahrrädern auch in der Nähe befindliche Stationen in Betracht zu ziehen, um so in selbstorganisierender Art und Weise eine erhöhte Verfügbarkeit von Fahrrädern an den Ausleihstationen zu erzielen. Dazu wurde ein dezentraler Koordinationsansatz verwendet, um die alternativen Ausleih- und Abgabestationen zu bestimmen. Hierfür wurde ein Koordinationsprozess realisiert, mit dem die Stationen periodisch andere Stationen innerhalb eines bestimmten Kommunikationsradius über ihre aktuelle Auslastung informieren. Stationen sammeln diese Benachrichtigungen ein und nutzen sie, um die alternativen Ausleih- und Rückgabestationen zu bestimmen, um somit eine möglichst gleichmäßige Auslastung der Stationen zu erreichen. Dabei wurde anhand des Beispiel sowohl die Realisierung des Koordinationsprozesses als auch der Integrationsansatz des entwickelten Frameworks beschrieben. Ferner wurde der Einfluss unterschiedlicher Kommunikationsradien als Koordinationsparameter auf die Auslastungsraten der Stationen untersucht.

Außerdem wurde beschrieben, wie das Konzept der strukturellen Adaption von Koordinationsprozessen dazu verwendet werden kann, um dynamisch zur Laufzeit zwischen einer LKW-basierten Umverteilung der Fahrräder und der Benutzer-basierten Umverteilung umzuschalten. Somit wurde es möglich, zunächst die benutzerfreundlichere Umverteilung der Fahrräder auf Basis von LKW-Transporten zu verwenden und erst auf die Einbeziehung der Benutzer umzuschalten, wenn die Auslastungsraten der Stationen nicht mehr optimal sind. Dabei wurde gezeigt, wie das Konzept der strukturellen Adaptionen von Koordinationsprozessen für das entwickelte generische Koordinationsframework angepasst wurde. Anhand dieser Anwendungsstudie konnte somit im Sinne einer Validierung gezeigt werden, dass das entwickelte Koordinationsframework geeignet ist, um dezentrale Koordinationsprozessen zu realisieren und auf Basis dieser ein selbstorganisierendes Verhalten umzusetzen.

Im zweiten Teil dieses Kapitels erfolgte die Bewertung der im Rahmen dieser Dissertation erzielten neuen Erkenntnisse. Dazu wurde zunächst das entwickelte dezentrale Koordinationsframework mit existierenden Ansätzen, die eine ähnliche Zielsetzung verfolgen, verglichen. Dafür wurde eine Klassifizierungsmatrix erstellt, anhand derer die existierenden Ansätze mit dem entwickelten Framework verglichen wurden. Dabei zeigte sich insbesondere, dass der entwickelte Ansatz eine konsequente Weiterentwicklung der Vorarbeiten

(*DeCoMAS* und *Coordination Space*) ist und das ursprüngliche Konzept der Realisierung von selbstorganisierenden Verhalten auf Basis von dezentraler Koordination sinnvoll durch eine Loslösung vom Agentenparadigma ergänzt. Dadurch wird es möglich, auch komplexe, verteilte Systeme, die aus unterschiedlichen heterogenen Softwarekomponenten bestehen, wie sie z.B. im Umfeld der Co-Simulationen eine Rolle spielen, zu koordinieren. Weiterhin wurde deutlich, dass der entwickelte Ansatz (und seine Vorarbeiten) sich von den existierenden Ansätzen primär in der Art der Koordinationsbeschreibung unterscheidet. Während die anderen Ansätze hier auf eine imperative Form der Koordinationsbeschreibung setzen, erfolgt diese im entwickelten Framework in deklarativer Form.

Die abschließende Bewertung erfolgte dann durch die Beantwortung der in der Einleitung formulierten Forschungsfragen. Hier wurden die minimalen Eigenschaften beschrieben, die generische Software-Komponenten aufweisen müssen, um koordinierbar zu sein und darauf aufbauend beschrieben, wie ein Rahmenwerk konstruiert sein muss, damit auf Basis dieser identifizierten Eigenschaften die Koordination erfolgen kann. Die Beantwortung dieser zwei Fragen orientierte sich dabei eng an den Konzepten und der Architektur des entwickelten Frameworks. Die dritte Forschungsfrage zielte schließlich darauf ab, wie ein einem selbstorganisierenden System dynamische Ineffizienzen in Form eines unerwünschten Verhaltens entgegengewirkt werden kann. Hier wurde das im Rahmen dieser Dissertation entwickelte Konzept der strukturellen Adaption von Koordinationsprozessen zur Beantwortung der Frage herangezogen und auf dessen Stärken und Schwächen eingegangen.

# 8. Zusammenfassung und Ausblick

In diesem Kapitel werden zunächst die Erkenntnisse und Ergebnisse dieser Arbeit zusammenfasst. Dabei wird zuerst ein Überblick über die Konzeption selbstorganisierender Systeme und deren Anwendung gegeben, wobei insbesondere das Paradigma der Selbstorganisation und dessen Umsetzung auf Basis von dezentraler Koordination im Fokus steht. Anschließend wird ausgehend von der Charakterisierung aktueller Herausforderungen auf dem Gebiet der systematischen Entwicklung selbstorganisierender Systeme und der existierenden Konstruktionsansätzen der Bedarf für die Lösungsansätze deutlich gemacht, die im Rahmen dieser Arbeit entwickelt wurden. Diese werden dabei kurz in zusammengefasster Form präsentiert, ehe schließlich ein Ausblick auf mögliche zukünftige Arbeiten im Umfeld dieser Arbeit gegeben wird.

## 8.1. Zusammenfassung

Aufgrund der zunehmenden Heterogenität, Dynamik, Vernetzung und Komplexität moderner IKT-Systeme wird die manuelle Konfiguration, Administration und Wartung solcher Systeme zunehmend schwieriger, was in einem erheblichen finanziellen und zeitlichen Mehraufwand resultiert. Folglich besteht ein Bedarf nach neuartigen Lösungsansätzen, um diese Herausforderungen zu bewältigen. Die Zielsetzung ist es also, IKT-Systeme zu entwickeln, die in der Lage sind, sich ohne manuelle Eingriffe selbstständig zu verwalten und an neue Situationen anzupassen. Diese Fähigkeit wird gemeinhin als *Selbstadaptivität* bezeichnet. Im Kontext verteilter Anwendungen und Systeme werden die unterschiedlichen Ausprägungen dieser Fähigkeit als *Self-** Eigenschaften bezeichnet, wobei der Stern ein Platzhalter für Eigenschaften wie z.B. Selbst-Konfiguration, Selbst-Optimierung und Selbst-Heilung ist. Grundsätzlich gehören Selbstadaptivität und die damit verbundenen Self-* Eigenschaften zu den wünschenswerten nichtfunktionalen Eigenschaften moderner, verteilter Anwendungen und umfassen dabei auch erstrebenswerte Eigenschaften wie Skalierbarkeit, Robustheit und Ausfallsicherheit.

Auf Grund des hohen Potenzials, das selbstadaptive Systeme bieten, gibt es eine Reihe von Forschungsinitiativen und -projekten, die sich mit der systematischen Entwicklung derartiger Systeme beschäftigten. Ein grundlegendes Unterscheidungsmerkmal besteht dabei darin, ob die Steuerung der Systeme auf einer zentralen Kontrollinstanz oder einem dezentralen Steuerungsansatz beruht. Dezentrale Steuerungsansätze haben dabei ihre Vorteile im Hinblick auf nichtfunktionale Eigenschaften wie Skalierbarkeit, Fehlertoleranz und Robustheit und sind somit besser geeignet, diesen nichtfunktionalen Anforderungen gerecht zu werden, die ihrerseits ein Schlüsselaspekt moderner, verteilter

Systeme sind. Innerhalb der dezentralen Lösungsansätze hat das Konzept der *Selbstorganisation* große Bedeutung erlangt und stellt damit auch einen zentralen Aspekt dieser Arbeit dar. Selbstorganisation beschreibt dabei einen dynamischen und adaptiven Prozess, der in Abwesenheit einer Kontrollinstanz auf Basis lokaler Interaktionen, die auf mikroskopischer Ebene stattfinden, in der Lage ist, auf makroskopischer Ebene Strukturen herauszubilden und diese im Hinblick auf sich verändernde Situationen anzupassen. Das Konzept der Selbstorganisation hat diese Eigenschaften sowohl in natürlichen als auch in technischen Domänen mehrfach erfolgreich nachgewiesen.

Die aufgeführten Eigenschaften selbstorganisierender Systeme machen deutlich, warum dieses Konzept in der Lage ist, den beschriebenen Herausforderungen moderner IKT-Systeme gerecht zu werden. Dazu ist es allerdings erforderlich, dass eine systematische softwaretechnische Nutzbarmachung gelingt. Diese gestaltet sich allerdings als herausfordernd im Hinblick auf die nichtlinearen Eigenschaften und das nichtdeterministische Verhalten, das selbstorganisierende Systeme aufweisen können sowie die mögliche Herausbildung emergenter Strukturen und ggf. daraus resultierenden, ungewollten emergenten Phänomen. Außerdem stellt das Fehlen von formalen Methoden zur Überprüfung und Analyse des Laufzeitverhaltens die systematische Entwicklung solcher Systeme vor zusätzliche Herausforderungen.

Damit die systematische softwaretechnische Nutzbarmachung von selbstorganisierenden Systemen gelingt, ist daher zunächst ein grundlegendes Verständnis solcher Systeme erforderlich. Das 2. Kapitel beschäftigt sich deswegen zunächst mit dieser Thematik und stellt dabei insbesondere den Bezug zu verwandten Aspekten wie Emergenz, Interaktion und Koordination her. Hier wird insbesondere die Bedeutung von Koordination für die Konzeption von selbstorganisierenden Systemen hervorgehoben, die entscheidend ist für die Interaktionen, die auf mikroskopischer Ebene abgebildet werden sollen und aus denen sich auf makroskopischer Ebene das selbstorganisierende Verhalten herausbildet. In Abwesenheit von zentralen Steuerungskomponenten können dezentrale Koordinationsmechanismen genutzt werden, um die Abhängigkeiten zwischen den verschiedenen Entitäten eines Systems zu verwalten und ihre Interaktionen untereinander abzubilden. Um das Verständnis und die Funktionsweise von Koordinationsmechanismen zu vertiefen, wurde dabei eine Klassifizierung vorgenommen, die eine Zuordnung von etablierten Koordinationsmustern auf typische Problemstellungen in verteilten Systemen vornimmt. Somit ist es möglich, anhand der Zuordnung, für typische Problemstellungen in verteilten Systemen die geeigneten Koordinationsmuster zu finden. Dadurch erfolgte eine erste Systematisierung des Konzepts der Selbstorganisation im Hinblick auf die Nutzbarmachung in verteilten Systemen. Weiterhin wurden im Rahmen des 2. Kapitels charakteristische Eigenschaften selbstorganisierender Systeme im Hinblick auf ihre Anwendung in verteilten Systemen diskutiert und einige Beispiele für verteilte Systeme, die von selbstorganisierenden Eigenschaften profitieren, präsentiert. Dadurch wurde das Potential

des Konzeptes der Selbstorganisation im Hinblick auf nichtfunktionale Eigenschaften verteilter Systeme verdeutlicht.

Ausgehend von dieser grundlegenden Betrachtung der Konzepte selbstorganisierende Systeme beschäftigte sich das 3. Kapitel mit einer weiterführenden Analyse der Hauptbestandteile solcher Systeme. Dabei wurden zunächst die Hauptbestandteile Feedback-Schleifen, Koordination, Agenten bzw. Aktive Komponenten und Umgebungen identifiziert. Dabei lässt sich das dynamische Verhalten selbstorganisierender Systeme, das schlussendlich zum selbstadaptiven Verhalten führt, maßgeblich mit Feedback-Schleifen abbilden und zielgerichtet beeinflussen. Da selbstorganisierende Systeme aus einer Vielzahl an autonomen Komponenten bestehen, bietet das Paradigma der Agenten bzw. der Aktiven Komponenten eine geeignete Möglichkeit zur Programmierung dieser Systeme. Umgebungen können dabei verwendet werden, um Koordinationsmedien zu unterstützen, die auf indirekter Kommunikation basieren, wie z.B. stigmergiebasierte Ansätze. Um selbstorganisierende Systeme nun softwaretechnisch nutzbar zu machen, ist es weiterhin notwendig, den Entwicklungsprozess zu betrachten. Dabei wurde im Rahmen des 3. Kapitels aufgezeigt, dass selbstorganisierende Systeme neben den bereits angesprochenen positiven Eigenschaften auch spezifische Charakteristika aufweisen, die die softwaretechnische Nutzbarmachung vor spezielle Herausforderungen stellt. Dazu gehören u.a. das nichtlineare und nichtdeterministische Verhalten, die Schwierigkeit der Herstellung der Beziehung von Parametern auf mikroskopischer Ebene und der daraus auf makroskopischer Ebene resultierenden Dynamik, Unzulänglichkeiten formaler Methoden zur Analyse des Laufzeitverhaltens und ungewollte, emergente Phänomene, z.B. in Form von unerwünschten Oszillationen oder Dead- und Livelocks. Folglich wurden eine Reihe existierender Ansätze, die sich mit der systematischen Entwicklung von selbstorganisierenden Systemen beschäftigten, vorgestellt und analysiert. Dabei wurde deutlich, dass die Mehrheit der untersuchten Ansätze einen relativ starken Fokus auf die frühen Phasen, insbesondere die Designphase, legen und wenig Anleitung für die spätere Implementierung geben. Vor allem werden kaum konkrete Empfehlungen geben, wie die Beziehung zwischen Kern- und Koordinationsfunktionalität zu gestalten ist. Von den untersuchten Ansätzen hebt sich lediglich der *SodekoVS*-Ansatz hervor, der explizit die systematische Konstruktion selbstorganisierender Systeme mit einer Referenzarchitektur adressiert.

Basierend auf diesen Erkenntnissen wurden im 4. Kapitel zunächst die spezifischen Herausforderungen für die Konstruktion selbstorganisierender Systeme herausgearbeitet. Dabei wurde erkennbar, dass hierbei besonders die Beziehung zwischen der Kern- und der Koordinationsfunktionalität bzw. deren möglichst loser Kopplung eine besondere Herausforderung darstellt. Nach einem Vergleich in Form einer Klassifizierung und anschließender Bewertung existierender Konstruktionsansätze wurde im Speziellen auf den Konstruktionsansatz *Coordination Space* eingegangen. Dieser implementiert die *SodekoVS*-Referenzarchitektur und liefert dabei eine Lösung zur Kon-

struktion selbstorganisierender Multiagentensysteme. Der Konstruktionsansatz bietet dabei eine Lösung, um dezentrale Koordinationsmechanismen auf Basis von Koordinationsprozessen zu realisieren. Dabei wird die Umgebung des Multiagentensystems verwendet, um hier auf Basis von deklarativ beschriebenen Koordinationsprozessen indirekte Kommunikationsmuster abzubilden. Der *Coordination Space* Ansatz bietet dazu ein konzeptionelles Integrationsmodell für Koordinationsprozesse und eine Ausführungsumgebung für die *Jadex*-Agentenplattform. Im Rahmen dieser Dissertation wurde dieses Konzept um ein Verteilungskonzept erweitert, das die Verteilung eines Koordinationsraumes über mehrere Plattformen und Netzwerkknoten hinweg ermöglicht. Ausgehend von dem Konstruktionsansatz des *Coordination Space* wurde im 4. Kapitel weiterhin ein Konzept zur strukturellen Adaption von Koordinationsprozessen präsentiert, um die dritte Forschungsfrage, die dieser Dissertation zugrunde liegt, zu beantworten:

3. Wie kann in einem selbstorganisierenden System dynamischen Ineffizienzen in Form eines unerwünschten Verhaltens entgegen gewirkt werden?

Selbstorganisierende Systeme können unter bestimmten Bedingungen ein unerwünschtes Verhalten, z.B. in Form eines Performanzabfalls, aufweisen. Ein vielversprechendes Konzept zur Überwindung solcher dynamischer Ineffizienzen ist der dynamische Austausch bzw. die Rekonfiguration der Koordinationsprozesse, die das selbstorganisierende Verhalten abbilden. Daher wurde im Rahmen dieser Arbeit das Konzept der strukturellen Adaption von Koordinationsprozessen erarbeitet, welches es ermöglicht, auf Basis von zur Designzeit vordefinierten möglichen Adaptionen die Struktur des selbstorganisierenden Verhaltens zur Laufzeit automatisiert anzupassen. Dazu wurde eine Referenzarchitektur und ein Entwicklungsansatz entworfen, die den selbstadaptiven strukturellen Austausch der selbstorganisierenden Koordinationsprozesse in Multiagentensystemen unterstützen. Durch Anpassungen an den Koordinationsprozessen ist es auf einer übergeordneten Ebene möglich, die Struktur des Selbstorganisationsprozesses anzupassen, indem dazu Teilstrukturen aktiviert bzw. deaktiviert werden. So kann einem unerwünschten selbstorganisierenden Verhalten durch ein künstlich erzeugtes selbstorganisierendes Phänomen entgegengewirkt werden. Konzeptionell ist der Ansatz dabei vergleichbar mit dem lokalen, reaktiven Planen aus dem BDI-Agenten Modell, wo ein einzelner Agent einen Plan basierend auf seinen lokalen Informationen und Alternativen auswählt. Der Ansatz der strukturellen Adaption beschäftigt sich dabei mit der Auswahl einer geeigneten, vordefinierten Anpassung, die auf anwendungsglobaler Ebene basierend auf einem verteilten Konsens der beteiligten Entitäten ausgewählt wird. Dazu wird dabei jeder Agent mit einem Sensor ausgestattet, der dedizierte Performanz-Indikatoren überwacht, die Aufschluss über die Performanz bzw. die Güte des selbstorganisierenden Verhaltens geben. Wenn der Agent auf Basis seiner lokalen Informationen bzw. Zustände einen entsprechenden Performanzabfall des selbstorga-

nisierenden Verhaltens detektiert, löst er den verteilten Adaptionsprozess aus. Dabei entscheiden die beteiligten Agenten strikt ausgehend von ihren lokalen Informationen bzw. Zuständen, ob sie der Adaption zustimmen. Die Konsensfindung erfolgt dementsprechend dezentral.

Als Teil eines iterativen Entwicklungsansatzes, der vielen selbstorganisierenden Systemen zugrunde liegt, spielen Simulationen eine wichtige Rolle, wenn es um die Validierung und Evaluation dieser Systeme geht. Da die Entwicklung selbstorganisierender Systeme von zahlreichen Herausforderungen begleitet ist, die sich erst zur Laufzeit herausbilden, hat sich ein iterativer Entwicklungsansatz etabliert, bei dem Simulationen genutzt werden, um zu testen, wie sich das entwickelte, selbstorganisierende System zur Laufzeit verhält. Zeigt das System dabei nicht das gewünschte Verhalten oder tritt z.B. unerwünschtes emergentes Verhalten auf, können im Zuge des iterativen Entwicklungsansatzes z.B. die Koordinationsprozesse, die für die Realisierung der selbstorganisierenden Dynamik verantwortlich sind, angepasst werden. Diese Anpassungen können dann wiederum mit Hilfe von Simulationen untersucht werden, um so zu evaluieren und zu validieren, dass sich das selbstorganisierende System wie gewünscht verhält. Simulationen sind aber nicht nur als Teil eines iterativen Vorgehens zu Validierungs- und Evaluationszwecken geeignet, sondern profitieren ihrerseits auch von selbstorganisierenden Eigenschaften oder der Fähigkeit zur Koordination. Daher wurden im 5. Kapitel skalierbare, verteilte und adaptive Simulationssysteme betrachtet. Von besonderem Interesse waren dabei Co-Simulationen. Bei einer Co-Simulation formen unterschiedliche Sub-Systeme gemeinsam ein gekoppeltes Problem. Dabei erfolgt sowohl die Modellierung als auch die Ausführung der Simulation verteilt. Unter einer verteilten Modellierung wird dabei in diesem Kontext die Verwendung von unterschiedlichen Werkzeugen, Programmen und Programmiersprachen verstanden. Während der verteilten Ausführung interagieren die einzelnen Sub-Simulationen, indem sie Daten untereinander austauschen. Eine der Hauptherausforderungen bei der Realisierung solcher Co-Simulationssysteme ist dementsprechend die sinnvolle Orchestration bzw. Koordination dieser. Aufgrund der Heterogenität der verwendeten Technologien im Umfeld der Co-Simulationen wird hierbei ein Konstruktionsansatz benötigt, der, im Gegensatz zu Ansätzen wie dem *Coordination Space* möglichst technologieunabhängig ist, die Koordination von generischen Softwarekomponenten unterstützt und sich nicht auf ein bestimmtes Architekturparadigma, wie z.B. das Agentenparadigma, beschränkt.

Im 6. Kapitel wurde dementsprechend ein neuartiger Ansatz zur Realisierung dezentraler Koordination in verteilten Systemen vorgestellt, der auf die Anforderungen eingeht, die entstehen, wenn heterogene Systeme wie z.B. Co-Simulationssysteme koordiniert werden sollen. Der im Rahmen dieser Arbeit entwickelte, neuartige Ansatz greift dabei einige Konzepte aus der Integrationsarchitektur des *Coordination Space* Konzepts bzw. dessen Vorarbeiten auf und adaptiert diese zu einem allgemeineren Koordinationskonzept, das die Konstruktion selbstorganisierender Systeme unabhängig von der zugrun-

deliegenden Systemarchitektur erlaubt. Dabei wurden im 6. Kapitel insbesondere, die ersten beiden Forschungsfragen beantwortet, die dieser Dissertation zugrunde liegen:

1. Welche Eigenschaften müssen generische Software-Komponenten aufweisen, um koordinierbar zu sein?

2. Wie muss ein Rahmenwerk konstruiert werden, damit auf Grundlage dieser identifizierten Eigenschaften generische Software-Komponenten koordiniert werden können?

Grundsätzlich lässt sich hierbei im Bezug auf die erste Forschungsfrage feststellen, dass es einen Unterschied zwischen der Koordination von Komponenten gibt, die im Rahmen einer Ausführungsumgebung verwaltet werden und solchen, die nicht verwaltet, also unabhängig von einer speziellen Ausführungsumgebung ausgeführt werden. Verwaltete Komponenten wie z.B. Software-Agenten bieten dabei spezielle Modellierungslemente, mit denen zur Entwicklungszeit das Verhalten der Komponenten bestimmt werden kann und die zur Laufzeit observiert werden können, um den Zustand der Komponente zu bestimmten. Somit ist es möglich, Zustände oder Ereignisse, die für die Koordination relevant sind, zu observieren und zu benutzen, um Koordinationsprozesse auszulösen bzw. durch diese das Verhalten der Komponenten zu beeinflussen. Bei nicht verwalteten Komponenten fehlt es an solchen implizit vorhandenen Modellierungselementen. Daher wurde in Kapitel 6 das Konzept expliziter Koordinationsereignisse präsentiert. Hierbei handelt es sich um Ereignisse, die von den entsprechenden Komponenten ausgelöst werden, sobald ein für die Koordination relevanter Zustand erreicht wird. Durch die Einführung dieser Elemente wird allerdings die strikte Trennung zwischen Koordinations- und Anwendungslogik, wie sie von [GC92] gefordert wird, etwas aufgeweicht. Allerdings liefert das im 6. Kapitel präsentierte Koordinationsrahmenwerk an dieser Stelle ein Konzept, das eine möglichst lose Kopplung zwischen den zu koordinierenden Komponenten und den sogenannten Endpunkten, die die Komponenten zum Zwecke der Koordination überwachen und beeinflussen, ermöglicht. Die Komponenten benötigen damit kein explizites Wissen darüber, ob sie koordiniert werden oder nicht. Zwar müssen sie explizit Koordinationsereignisse auslösen, deren mögliche Weiterverarbeitung in eventuell vorhandenen Endpunkten geschieht aber transparent für die Komponenten. Somit ist die strikte Trennung zwischen Anwendungs- und Koordinationslogik zwar etwas aufgeweicht, aber eine lose Kopplung zwischen diesen beiden Aspekten besteht noch immer. Gleichzeitig erlaubt es der Ansatz auf diese Weise auch Systeme zu koordinieren, die aus heterogenen Software-Komponenten bestehen.

Im 7. Kapitel erfolgte eine Validierung der in dieser Arbeit entwickelten Lösungen zur strukturellen Adaption von Koordinationsprozessen sowie dem entwickelten dezentralen Koordinationsrahmenwerk. Dabei wurde anhand ausgewählter Anwendungsstudien die Eignung der entwickelten Lösungen

bezogen auf ihren Einsatzzweck hin überprüft. Hierbei wurde sowohl auf konzeptioneller wie auch auf technischer Ebene, die praktische Nutzbarkeit der Lösungsansätze aufgezeigt. Weiterhin erfolgte eine Bewertung der eigenen Ansätze im Vergleich mit existierenden Ansätzen, wobei eine Abgrenzung der entwickelten von den existierenden Ansätzen durchgeführt wurde. Weiterhin wurden im 7. Kapitel die im Rahmen dieser Dissertation entwickelten Lösungskonzepte im Hinblick auf die in der Einleitung formulierten Forschungsfragen bewertet.

Zusammenfassend wurde einerseits mit dem Konzept der strukturellen Adaption von Koordinationsprozessen ein Ansatz präsentiert, der es ermöglicht, unerwünschtem Verhalten in selbstorganisierenden Systemen durch antizipierte Anpassungen an der Struktur, der dem selbstorganisierenden Verhalten zugrundeliegenden Koordinationsprozesse, entgegenzuwirken. Weiterhin wurde ein neuartiger Ansatz zur dezentralen Koordination in verteilten Systemen erarbeitet, der auf die Anforderungen eingeht, die entstehen wenn heterogene Systeme koordiniert werden sollen, und somit die Koordination von generischen Software-Komponenten ermöglicht.

## 8.2. Ausblick

In diesem Abschnitt wird ein kurzer Überblick über mögliche, zukünftige Arbeiten im Umfeld dieser Dissertation gegeben.

### 8.2.1. Dezentrale Koordination von Co-Simulationen

Um das im Rahmen dieser Dissertation entwickelte dezentrale Koordinationsrahmenwerk weiterführend zu validieren und so seine Eignung zu prüfen, soll es im Rahmen des *GEWISS* Projektes eingesetzt werden, um die Ausführung unterschiedlicher Simulationssysteme im Rahmen einer Co-Simulation zu koordinieren. Im Rahmen dieses Projekts soll ein Geographisches Wärme Informations- und Simulationssystem für die Freie und Hansestadt Hamburg entwickelt werden. Um Klimaschutzpotentiale auszureizen und die vorhandenen Ressourcen mit hoher Kosteneffektivität einzusetzen, ist es notwendig, die strategische Wärmeplanung mit der Stadtentwicklung zu verzahnen. Dabei ist eine Abstrahierung von der räumlichen Verortung des Gebäudebestands offensichtlich nicht möglich. Aspekte wie die Konversion von städtischen Flächen, Nachverdichtung, Umnutzung oder auch Abriss und Sanierung von Gebäuden bzw. Gebäudeensembles sollten dazu auf die lokal vorhandenen Wärmequellen abgestimmt werden. Im Umkehrschluss bedeutet dies, dass insbesondere die netzgebundene Wärmeversorgung mit Bezug auf die aktuellen und zukünftigen Gegebenheiten des Gebäudebestands vor Ort geplant und fortentwickelt werden sollte. Für diese Verzahnung ist es nötig, dass Daten und Analysen mit räumlichem Bezug erhoben werden. Die Verknüpfung dieser Daten und Analysen zu Wärmebedarfen und Gebäudeentwicklung sollte in einem geographischen Informationssystem (GIS) erfolgen.

Das Ziel dieses Projekt ist es, ein solches GIS erweitert, um geographische Wärmeinformationen (GEWIS) und Simulationen für zukünftige Entwicklung bereitzustellen (GEWISS). Daher gilt es, ein Simulationssystem zu entwickeln, mit dem Anwender mittel- und langfristige Entwicklung des Wärmeversorgungssystems ebenso wie mögliche Stadtentwicklungsszenarien simulieren können. Dabei sollen sowohl externe Rahmenbedingungen (z.B. die Entwicklung von Energiepreisen) als auch kommunale Politikmaßnahmen (z.B. Vorschriften oder finanzielle Anreize für bestimmte bauliche Maßnahmen) vorgegeben und in ihrer Wirkung untersucht werden können. Dies geschieht mit dem Ziel, die Konzeption von Geschäftsmodellen ebenso wie unterschiedliche stadtplanerische und bauliche Maßnahmen systematisch explorieren zu können. Dadurch soll es möglich sein, genau abgestimmte Maßnahmen zu finden, die der Stadt Hamburg ein bestmögliches Erreichen von Klimaschutzzielen im Wärmesektor erlauben. Aus dem Gesamtziel des Vorhabens ergeben sich eine Reihe von wissenschaftlichen und technischen Rahmenbedingungen, die den Einsatz unterschiedlicher Simulationssysteme im Sinne einer Co-Simulation erfordern. Um sowohl die Ausführung dieser unterschiedlichen Simulationssysteme als auch den Austausch von Daten zwischen ihnen zu koordinieren, erscheint die Verwendung des im Rahmen dieser Arbeit entwickelten Koordinationsrahmenwerks sinnvoll.

### 8.2.2. Strukturelle Adaption von Koordinationsprozessen

Das Konzept der strukturellen Adaption von Koordinationsprozessen, wie es im Rahmen dieser Dissertation präsentiert wurde, erfordert ein dynamisches Benchmarking der Performanz des realisierten selbstorganisierenden Verhaltens. Aktuell erfolgt die Bewertung der selbstorganisierenden Dynamik dabei über lokale Performanz-Indikatoren, die aus Sicht einer einzelnen Komponente Ausdruck darüber geben, wie gut die selbstorganisierende Dynamik für diese Komponente funktioniert. Dann wird basierend auf einem verteilten Wahlalgorithmus ein globaler Konsens über die mögliche Ausführung der Adaption gefunden, wobei die einzelnen Komponenten auf Basis ihres lokalen Zustandes abstimmen. Bessere bzw. genauere Benchmarking-Ergebnisse könnten dabei erzielt werden wenn nicht nur eine lokale Betrachtung von Performanz-Kriterien in den Komponenten auf mikroskopischer Ebene erfolgt, sondern eine mesoskopische Beobachtungsschicht eingeführt wird, welche die Zustände einer Menge von Komponenten betrachtet und auswertet. Die Idee dabei ist es, spezielle Beobachtungskomponenten im System einzuführen, die jeweils eine Teilmenge der teilnehmenden Komponenten überwachen und versuchen, für diese Teilmenge festzustellen, ob ein defizitärer Zustand im Hinblick auf die selbstorganisierende Dynamik besteht. Diese Beobachtungskomponenten könnten dann in ähnlicher Art und Weise wie die lokalen Komponenten bisher ein verteiltes Konsensverfahren nutzen, um sich über mögliche Adaptionen abzustimmen, wenn sie in ihrer beobachtenden Teilmenge von Komponenten ein defizitäres Verhalten fest-

stellen. Dadurch, dass diese Beobachtungskomponenten nicht nur die lokale Sicht einer einzelnen Komponente auf die Qualität der selbstorganisierenden Dynamik, sondern Überblick über die Zustände einer Menge von Komponenten haben, sollten sie bessere Aussagen über die aktuelle Qualität der selbstorganisierenden Dynamik treffen können.

Eine weitere mögliche Erweiterung des Konzepts der strukturellen Adaptionen von Koordinationsprozessen wäre es, zu versuchen, vor der Durchführung einer strukturellen Adaption abzuschätzen, was der Einfluss dieser auf die Qualität der selbstorganisierenden Dynamik ist und sie mit einer entsprechenden Kennzahl zu versehen. Bestehen mehrere Adaptionsalternativen, so können diese auf Basis der Einschätzungen verglichen werden, um so die Effektivste auszuwählen. Hierbei wären allerdings umfangreiche Arbeiten im Umfeld der Einschätzung der Qualität von selbstorganisierenden Dynamiken erforderlich. Gerade im Umfeld von selbstorganisierenden Systemen mit sich verändernden Umgebungen sowie nichtlinearen und nichtdeterministischen Verhalten erscheint dies eine sehr anspruchsvolle Aufgabe zu sein. Eine Möglichkeit, um diese Komplexität zu handhaben, wäre es, im Vorfeld Simulationen zu verwenden, um den Einfluss der unterschiedlichen Koordinationsprozesse zu messen. Dadurch könnte im Vorfeld ein Katalog angelegt werden, der als Ergebnis dieser Simulationen Informationen darüber enthält, wie gut ein bestimmter Koordinationsprozess unter welchen Rahmenbedingungen funktioniert. Im Falle eines defizitären Verhaltens, das eine strukturelle Adaption notwendig macht, müssten dann nur die entsprechenden Rahmenbedingungen bestimmt und es könnte im Katalog nachgeschaut werden, welche Koordinationsprozesskombination für diesen Fall optimal ist.

### 8.2.3. Policen-basierter Ansatz zur Einhaltung nichtfunktionaler Anforderungen

Eine verwandte Forschungsidee, die ursprünglich in Rahmen von [Pre13] angedacht wurde, ist ein Policen-basierter Ansatz zur Durchsetzung von nichtfunktionalen Anforderungen in selbstadaptiven und selbstorganisierenden Systemen. Bei [Pre13] handelt es sich bisher nur um eine konzeptionelle Idee, die nicht weiterführend auf ihre Machbarkeit hin untersucht wurde.

In aktuellen, komplexen IKT-Infrastrukturen ist es üblich, dass unterschiedliche Systeme bzw. Anwendungen auf gemeinsame Ressourcen zurückgreifen, ohne dass sie gemeinsame funktionale Anforderungen besitzen. In [Pre13] wurde daher eine neue Sichtweise auf die Modellierung von verteilten Systeme vorgeschlagen, bei der diese unterschiedlichen, verteilten Anwendungen als ein gemeinsames verteiltes System betrachtet werden, sofern sie gleichzeitig auf die selben Ressourcen zugreifen. Betrachtet man ein solches verteiltes System, das aus unterschiedlichen Anwendungen besteht, so bieten sich viele Möglichkeiten, das Verhalten der unterschiedlichen Anwendungen anzupassen, um sicherzustellen, dass auch bei plötzlich auftretenden

Fehlerfällen und sich verändernden Bedingungen alle Anwendungen ihre nichtfunktionalen Anforderungen erfüllen können.

Abbildung 8.1 zeigt eine konzeptionelle Architektur und einen Entwicklungsansatz, die dieses Konzept auf Basis von zu spezifizierenden Policen theoretisch umsetzen können. Ein System-Designer spezifiziert dabei die nichtfunktionalen Anforderungen der unterschiedlichen Anwendungen in Form von Policen. Eine spezielle *Transform* Komponente, die mit Adaptern für unterschiedliche Sprachen ausgestattet ist (*Language Adapters)*, transformiert diese in maschinenlesbare Anforderungspolicen (*Requirement Policies*). Ein System-Operator, der über spezifisches Wissen über die Infrastruktur und die vorhandenen Ressourcen verfügt, spezifiziert sogenannte *Adaptation Rules*, die mögliche Anpassungen an den Anwendungen beinhalten. Diese werden ebenfalls in maschinenlesbare Anweisungen transformiert. Diese Adaptionsanweisungen sowie die Policen werden in einer dezentralen Wissensbasis des verteilten Systems gespeichert (*Knowledge and Policy Base*). Da die Policen sicherstellen sollen, dass die nichtfunktionalen Anforderungen erfüllt werden, enthalten diese Bedingungen, die herangezogen werden, um die entsprechenden Adaptionsanweisungen auszuwählen. Diese Bedingungen hängen dabei von Parametern wie z.B. der Systemgröße ab. Dabei ist es denkbar, dass für Systeme mit geringer Größe zentralisierte, selbstadaptive Anpassungen besser geeignet sind, um das System zu rekonfigurieren. Im Fall von größeren, komplexen Systemen mag eine zentralisierte Lösung nicht ausreichend gut skalieren, sodass in diesem Fall eine selbstorganisierende Anpassung besser geeignet ist. Die in Abbildung 8.1 angedachte Kontrollmiddleware wählt die möglichen Anpassungsanweisungen anhand der spezifizierten Policen aus und führte diese dann aus, um die Einhaltung der nichtfunktionalen Anforderungen sicherzustellen. Dazu ist es notwendig, dass die Systeme, die kontrolliert werden (*Systems under Control*), mit entsprechenden Sensoren und Effektoren ausgestattet sind. Die Sensoren sind dabei als dezentrale, verteilte Monitore konzipiert, die die Einhaltung der Policen überwachen und im Fall einer Verletzung dieser die entsprechenden Anpassungsanweisungen auswählen, um das System zu rekonfigurieren. Die Effektoren stellen Schnittstellen bereit, die von den Kontrollprozessen verwendet werden können, um die Systementitäten zu beeinflussen, um so die Einhaltung der Policen sicherzustellen. Eine verteilte Komponente zur Ablaufverfolgung (*Trace*) erlaubt es dem Operator dabei, den Zustand des Systems und all die ausführten Adaptionsanweisungen, die von der Middleware durchgeführt wurden, zu überwachen. Auf Basis dieser Informationen ist der Operator damit in der Lage, die Effekte der Adaptionsanweisungen nachzuvollziehen und kann diese wenn nötig ggf. anpassen.

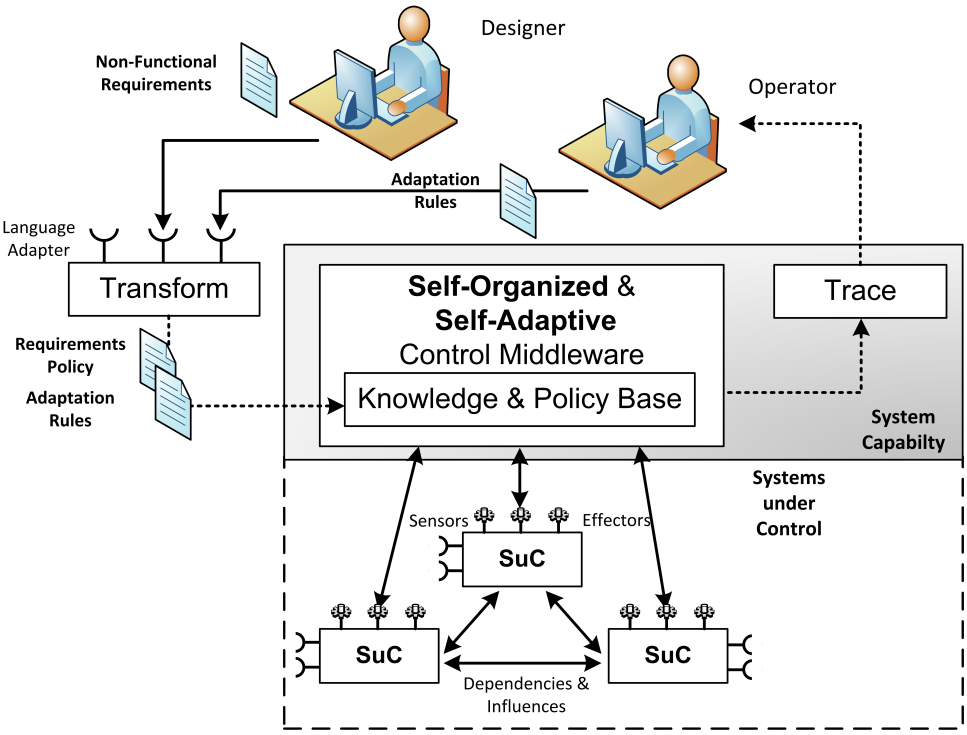



**Abbildung 8.1.:** Konzeptionelle Architektur und Entwicklungsansatz des Policen-basierten Ansatzes zur Einhaltung von nichtfunktionalen Anforderungen in selbstadaptiven und selbstorganisierenden Systemen - aus (Pre13, S. 4).

# A. Quelltextauszüge

```
1 <xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
       xmlns:xs="http://www.w3.org/2001/XMLSchema">
2   <xs:element name="struct_adapt">
3     <xs:complexType>
4       <xs:sequence>
5         <xs:element name="agents">
6           <xs:complexType>
7             <xs:sequence>
8               <xs:element name="agent" maxOccurs="unbounded" minOccurs="0">
9                 <xs:complexType>
10                  <xs:simpleContent>
11                    <xs:extension base="xs:string">
12                      <xs:attribute type="xs:string" name="id" use="optional
                            "/>
13                    </xs:extension>
14                  </xs:simpleContent>
15                </xs:complexType>
16              </xs:element>
17            </xs:sequence>
18          </xs:complexType>
19        </xs:element>
20        <xs:element name="adaptations">
21          <xs:complexType>
22            <xs:sequence>
23              <xs:element name="adaptation">
24                <xs:complexType>
25                  <xs:sequence>
26                    <xs:element name="realizations">
27                      <xs:complexType>
28                        <xs:sequence>
29                          <xs:element name="realization" maxOccurs="
                                unbounded" minOccurs="0">
30                            <xs:complexType>
31                              <xs:simpleContent>
32                                <xs:extension base="xs:string">
33        <xs:attribute type="xs:string" name="id" use="optional"/>
34        <xs:attribute type="xs:string" name="activate" use="optional"/>
35                                </xs:extension>
36                              </xs:simpleContent>
37                            </xs:complexType>
38                          </xs:element>
39                        </xs:sequence>
40                      </xs:complexType>
41                    </xs:element>
42                    <xs:element name="constraints">
43                      <xs:complexType>
44                        <xs:sequence>
45                          <xs:element name="constraint" maxOccurs="unbounded
                                " minOccurs="0">
46                            <xs:complexType>
47                              <xs:simpleContent>
48                                <xs:extension base="xs:string">
49        <xs:attribute type="xs:string" name="agent_id" use="optional"/>
50        <xs:attribute type="xs:string" name="element" use="optional"/>
51        <xs:attribute type="xs:string" name="type" use="optional"/>
52        <xs:attribute type="xs:byte" name="condition" use="optional"/>
53        <xs:attribute type="xs:byte" name="threshold" use="optional"/>
54                                </xs:extension>
```

```
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
    <xs:attribute type="xs:string" name="id"/>
    <xs:attribute type="xs:byte" name="answers"/>
    <xs:attribute type="xs:float" name="quorum"/>
    <xs:attribute type="xs:short" name="startDelay"/>
    <xs:attribute type="xs:string" name="reset"/>
    <xs:attribute type="xs:string" name="single"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

**Quelltext A.1:** XML Schema Definition (XSD) welche die Adaptionserweiterung der *MASDynamics* (SR09b) Sprache zeigt.

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
      xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="coordination-description">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="role-definitions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="role">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="agents">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="agent">
                            <xs:complexType>
                              <xs:simpleContent>
                                <xs:extension base="xs:string">
                                  <xs:attribute type="xs:string" name="class
                                       "/>
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute type="xs:string" name="name"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="deco-link-definitions">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="deco-link" maxOccurs="unbounded" minOccurs="
                   0">
                <xs:complexType>
                  <xs:sequence>
```

```
39                      <xs:element name="from">
40                        <xs:complexType>
41                          <xs:simpleContent>
42                            <xs:extension base="xs:string">
43                              <xs:attribute type="xs:string" name="role" use="
                                    optional"/>
44                              <xs:attribute type="xs:string" name="step" use="
                                    optional"/>
45                              <xs:attribute type="xs:string" name="field" use=
                                    "optional"/>
46                            </xs:extension>
47                          </xs:simpleContent>
48                        </xs:complexType>
49                      </xs:element>
50                      <xs:element name="selector">
51                        <xs:complexType>
52                          <xs:simpleContent>
53                            <xs:extension base="xs:string">
54                              <xs:attribute type="xs:string" name="class" use=
                                    "optional"/>
55                            </xs:extension>
56                          </xs:simpleContent>
57                        </xs:complexType>
58                      </xs:element>
59                      <xs:element name="to">
60                        <xs:complexType>
61                          <xs:simpleContent>
62                            <xs:extension base="xs:string">
63                              <xs:attribute type="xs:string" name="role" use="
                                    optional"/>
64                              <xs:attribute type="xs:string" name="step" use="
                                    optional"/>
65                              <xs:attribute type="xs:string" name="field" use=
                                    "optional"/>
66                            </xs:extension>
67                          </xs:simpleContent>
68                        </xs:complexType>
69                      </xs:element>
70                    </xs:sequence>
71                    <xs:attribute type="xs:string" name="id" use="optional"/>
72                  </xs:complexType>
73                </xs:element>
74              </xs:sequence>
75            </xs:complexType>
76          </xs:element>
77        </xs:sequence>
78        <xs:attribute type="xs:string" name="context"/>
79      </xs:complexType>
80    </xs:element>
81  </xs:schema>
```

**Quelltext A.2:** XML Schema Definition (XSD) zur Beschreibung von Koordinationsprozessen in einem verteilen System basierend auf Aktiven Komponenten.

# Eigene Veröffentlichungen

Folgende Veröffentlichungen sind aus den Ergebnissen im Umfeld dieses Dissertationsprojekts hervorgegangen:

SUDEIKAT, JAN, JAN-PHILIPP STEGHÖFER, HELLA SEEBACH, WOLFGANG REIF, WOLFGANG RENZ, THOMAS PREISLER und PETER SALCHOW: *Design and Simulation of a Wave-like Self-Organization Strategy for Resource-Flow Systems*. In: BOISSIER, OLIVIER, AMAL EL FALLAH-SEGHROUCHNI, SALIMA HASSAS und NICOLAS MAUDET (Herausgeber): *Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), Lyon, France, August 30 - September 2, 2010*, Band 627 der Reihe *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

SUDEIKAT, JAN, JAN-PHILIPP STEGHÖFER, HELLA SEEBACH, WOLFGANG REIF, WOLFGANG RENZ, THOMAS PREISLER und PETER SALCHOW: *A Wave-like Decentralized Reconfiguration Strategy for Self-organizing Resource-Flow Systems (Short Paper)*. Electronic Proceedings in Theoretical Computer Science, 27:32–33, 2010. Proceedings of the Workshop.

SUDEIKAT, JAN, JAN-PHILIPP STEGHÖFER, HELLA SEEBACH, WOLFGANG REIF, WOLFGANG RENZ, THOMAS PREISLER und PETER SALCHOW: *On the Combination of Top-down and Bottom-up Methodologies for the Design of Coordination Mechanisms in Self-organising Systems*. Information and Software Technology, 54(6):593–607, Juni 2012.

PREISLER, THOMAS und WOLFGANG RENZ: *Scalability and Robustness Analysis of a Multi-Agent based Self-healing Resource-flow System*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Federated Conference on Computer Science and Information Systems - FedCSIS 2012, Wroclaw, Poland, 9-12 September 2012, Proceedings*, Seiten 1261–1268, 2012.

RENZ, WOLFGANG, THOMAS PREISLER und JAN SUDEIKAT: *Mesoscopic Stochastic Models for Validating Self-Organizing Multi-Agent Systems*. In: *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, Seiten 119–126. IEEE Computer Society, 2012.

PREISLER, THOMAS: *Policy-Based Approach for Non-Functional Requirements in Self-adaptive and Self-Organizing Systems*. In: KLUSCH, MATTHIAS,

MATTHIAS THIMM und MARCIN PAPRZYCKI (Herausgeber): *Multiagent System Technologies - 11th German Conference, MATES 2013, Koblenz, Germany, September 16-20, 2013. Proceedings*, Band 8076 der Reihe *Lecture Notes in Computer Science*, Seiten 420–423. Springer, 2013.

PREISLER, THOMAS, WOLFGANG RENZ und ANTE VILENICA: *Bike-sharing System Reliability Problems - A Data-based Analysis and Simulation Architecture*. In: WOLFGANG KERSTEN, THORSTEN BLECKER, CHRISTIAN M. RINGLE (Herausgeber): *Sustainability and Collaboration in Supply Chain Management: A Comprehensive Insight into Current Management Approaches*, Band 16 der Reihe *Supply Chain, Logistics and Operations Management*, Seiten 83–97. Josef Eul Verlag, 8 2013.

PREISLER, THOMAS, ANTE VILENICA und WOLFGANG RENZ: *Decentralized Coordination in Self-Organizing Systems based on Peer-to-Peer Coordination Spaces*. Electronic Communications of the EASST, 56, 2013.

DETHLEFS, TIM, DIRK BEEWEN, THOMAS PREISLER und WOLFGANG RENZ: *A consumer-orientated Architecture for distributed Demand-Side-Optimization*. In: GÓMEZ, JORGE MARX et al. [GSV+14], Seiten 349–356.

DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *Multi-Agent-based Distributed Optimization for Demand-Side-Management Applications*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014.*, Seiten 1489–1496, 2014.

PREISLER, THOMAS, GREGOR BALTHASAR, TIM DETHLEFS und WOLFGANG RENZ: *Scalable Integration of 4GL-Models and Algorithms for massive Smart Grid Simulations and Applications*. In: GÓMEZ, JORGE MARX et al. [GSV+14], Seiten 341–348.

PREISLER, THOMAS, GREGOR BALTHASAR, TIM DETHLEFS und WOLFGANG RENZ: *Servicekomponenten-basierte Architektur für mikroskopische und makroskopische Simulation der städtischen Energieversorgung*. In: *VDE-Kongress 2014 Smart Cities, Frankfurt am Main, Deutschland*, 2014.

DETHLEFS, TIM, CHRISTOPH BRUNNER, THOMAS PREISLER, OLIVER RENKE, WOLFGANG RENZ und ANDREA SCHRÖDER: *Energy Service Description for Capabilities of Distributed Energy Resources*. In: GOTTWALT, SEBASTIAN, LUKAS KÖNIG und HARTMUT SCHMECK (Herausgeber): *Energy Informatics - 4th D-A-CH Conference, EI 2015, Karlsruhe, Germany, November 12-13, 2015, Proceedings*, Band 9424 der Reihe *Lecture Notes in Computer Science*, Seiten 24–35. Springer, 2015.

DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *Ant-Colony based Self-Optimization for Demand-Side-Management*. In: *SmartER Europe 2015, Essen, Germany*, 2015.

DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *A DER Registry System as an Infrastructural Component for future Smart Grid Applications*. In: *ETG Congress 2015 - Die Energiewende, Bonn, Deutschland, 17.-18. November*, 2015.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Data-Adaptive Simulation: Cooperativeness of Users in Bike-Sharing Systems*. In: KERSTEN, WOLFGANG, THORSTEN BLECKER und CHRISTIAN M. RINGLE (Herausgeber): *Proceedings of the Hamburg International Conference of Logistics*, Band 20. epubli GmbH, 2015.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Middleware for Constructing Decentralized Control in Self-Organizing Systems*. In: *2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015*, Seiten 325–330. IEEE, 2015.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Simulation as a service: A design approach for large-scale energy network simulations*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015*, Seiten 1765–1772. IEEE, 2015.

PREISLER, THOMAS und WOLFGANG RENZ: *Structural Adaptations for Self-Organizing Multi-Agent Systems*. In: *Seventh International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE), Nice, France*, 2015.

PREISLER, THOMAS, WOLFGANG RENZ und TIM DETHLEFS: *Structural Adaptation for Self-Organizing Multi-Agent Systems: Engineering and Evaluation*. International Journal on Advances in Intelligent Systems, 8(3&4):413–425, 2015.

DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *Dynamic Aggregation of Virtual Power Plants with a Registry System for Distributed Energy Resources*. In: *SmartER Europe 2016, Essen, Germany*, 2016.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *DeCoF: A Decentralized Coordination Framework for Various Multi-Agent Systems*. In: KLUSCH, MATTHIAS, RAINER UNLAND, ONN SHEHORY, ALEXANDER

POKAHR und SEBASTIAN AHRNDT (Herausgeber): *Multiagent System Technologies - 14th German Conference, MATES 2016, Klagenfurt, Österreich, September 27-30, 2016. Proceedings*, Band 9872 der Reihe *Lecture Notes in Computer Science*, Seiten 73–88. Springer, 2016.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Self-Organizing Redistribution of Bicycles in a Bike-Sharing System based on Decentralized Control*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.*, Seiten 1471–1480, 2016.

PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Structural Adaptations of Decentralized Coordination Processes in Self-Organizing Systems*. In: KOUNEV, SAMUEL, HOLGER GIESE und JIE LIU (Herausgeber): *2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016*, Seiten 263–268. IEEE, 2016.

# Abbildungsverzeichnis


2.1. Zusammenhang zwischen Emergenz und Selbstorganisation: (a) Selbstorganisation ohne Emergenz. (b) Emergenz ohne Selbstorganisation. (c) Emergenz und Selbstorganisation - aus [WH04, S. 10]. . . . . 17
2.2. Hierarchie adaptiver Systemeigenschaften - nach [ST09]. . . . . 21
2.3. Architektur eines verteilten Systems mit Ressourcen, die durch einen Autonomic-Manager auf Basis der MAPE-K Schleife gesteuert werden - aus [KC03, S. 44]. . . . . 26
2.4. Observer/Controller Architektur des *Organic Computing* - aus [BMMSP06, S. 186]. . . . . 27
2.5. Charakteristische Unterscheidungsmerkmale für verteilte Systeme mit der Fähigkeit zur Laufzeitanpassung - aus [Vil13, S. 26]. . . . . 28
2.6. Konzept von Adaptionsdynamik, Koordinationsprozess und Koordinationsmechanismus in einer verteilten Anwendung - nach [Sud10]. . . . . 29
2.7. Taxonomie unterschiedlicher Arten von Interaktion - nach [Pok07]. 31
2.8. Interaktionsmechanismen und assoziierte Selbstorganisationsmetaphern - aus [MMTZ06, S. 17]. . . . . 35
2.9. ISO 9126 Modell für Softwarequalität. . . . . 39
2.10. Fertigungsprozess eines Autos in einem Ressourcenflusssystem bestehend aus Robotern mit unterschiedlichen Fähigkeiten - aus [SSS+12]. . . . . 40
2.11. Beispielhafte Darstellung eines dezentralen Rekonfigurationsprozesses für ein Ressourcenflusssystem - nach [PR12]. . . . 43
2.12. Capital Bikeshare systemweite Anzahl von Fahrten von September 2010 bis März 2015 - aus [PDR15a, S. 8]. . . . . 46
2.13. Capital Bikeshare Umverteilungsmaßnahmen in 2014 - aus [PDR16b, S. 2]. . . . . 47
2.14. Anzahl der leeren und vollen Stationen in 2014 - aus [PDR16b, S. 2]. . . . . 47
2.15. UML Aktivitätsdiagramm das einen Ausleihvorgang mit vorgeschlagenen Alternativen aus Benutzersicht beschreibt - aus [PDR16b, S. 6]. . . . . 49
2.16. Auszug der simulierten Karte, die das Straßenmodell und einige der Stationen mit ihrem Kommunikationsradius zeigt - aus [PDR15a, S. 19]. . . . . 51
2.17. Ergebnisse der simulationsbasierten Auswertung der Ansätze - aus [PDR15a]. . . . . 52

2.18. Mittlere, relative Abweichung in der Anzahl der normalen Stationen im Vergleich zu dem Referenzszenario ohne selbstorganisierendes Verhalten - aus [PDR15a, S. 24]. . . . . 54
2.19. MAPE-K Schleife zur verteilten Planung und Optimierung in einer DSM Anwendung - aus [DBP+15, S. 3]. . . . . 56

3.1. CLD-Modell eines System mit einem Koordinationsmechanismus der auf digitalen Pheromonen basiert - aus [SR10b, S. 7]. . . 64
3.2. Taxonomie autonomer Agenten - nach [FG97]. . . . . 68
3.3. Paradigmen zur Entwicklung komplexer verteilter Systeme mit ihren jeweiligen Stärken und den typischen Anwendungsgebieten - aus [BP12b, Seite 3]. . . . . 70
3.4. Konzeptionelles Modell einer Aktiven Komponente - aus [BP12b, Seite 5]. . . . . 70
3.5. Anpassung der Entwicklungsmethode Unified Process für selbstorganisierende Systeme - nach [DWH05, S. 4]. . . . . 80
3.6. Experimentelle Methode zur zielgerichteten Entwicklung von selbstorganisierenden Systemen - nach [Edm05a, S. 94]. . . . . 82
3.7. SodekoVS Vorgehensmodell - nach [SBP+09, S. 7]. . . . . 84
3.8. SodekoVS Referenzarchitektur - nach [SBP+09, S. 6]. . . . . 85
3.9. Funktionsweise des Koordinationsendpunktes im SodekoVS-Ansatz - Ausschnitt aus [VSL+10, Abb. 1, S. 2]. . . . . 86

4.1. Konzeptionelles Modell des *Coordination Space* - aus [VSL+10]. . 97
4.2. Konzeptionelle Erweiterung der Integrationsarchitektur zu einem verteilten Coordinaton Space - aus [PVR13]. . . . . 100
4.3. Der Coordination Space und weitere Umgebungen als Teil einer Jadex-Anwendung - aus [Vil13, S. 98]. . . . . 103
4.4. Die Realisierung der verteilten Ausführungsumgebung des Coordination Space in Jadex - aus [PVR13, S. 8]. . . . . 107
4.5. Ein verteiltes Koordinationmedium basiered auf einem *Remote Service Call* Ansatz - aus [PVR13, S. 9]. . . . . 109
4.6. Ein verteiltes Koordinationmedium basiered auf einem *Publish/Subscribe* Ansatz - aus [PVR13, S. 9]. . . . . 110
4.7. Ein verteiltes Koordinationmedium basiered auf einem *Dishtributed Hash Table* Ansatz - aus [PVR13, S. 9]. . . . . 111
4.8. Einordnung des Ansatzes zur strukturellen Adaption von Koordinationsprozessen (roter Punkt) in die unterschiedlichen Dimensionen adaptiver Systeme - aus [PRD15, S. 1]. . . . . 114
4.9. Einordnung struktureller Adaptionsprozesse in dem Kontext eines selbstorganisierenden Multiagentensystems - nach [PRD15]. 115
4.10. Konzeptioneller Vergleich zwischen Planen in einem BDI-Agenten und der strukturellen Adaption. . . . . 117
4.11. Erweiterung der SodekoVS Referenzarchitektur um strukturelle Adaptionen (blaue Rechtecke) - nach [PRD15, S. 5]. . . . . 118

4.12. UML Aktivitätsdiagramm das den Adaptions- und Wahl-Algorithmus im Monitor (links) und Abstimmungsdienst (rechts) veranschaulicht - nach [PRD15]. . . . . 122

5.1. Zeithorizont und Planungsebenen in unterschiedlichen Simulationstypen im Bereich der Energieversorgung - aus [PBDR14b, S. 2]. . . . . 127
5.2. Servicekomponenten-basierte Architektur für ein adaptives Co-Simulationssystem - aus [PBDR14b, S. 5]. . . . . 130
5.3. Skalierbare Integration von MATLAB Code - aus [PBDR14a, S. 5]. 133
5.4. Integration einer skalierbaren 4GL-Komponente in die Architektur für ein servicebasiertes Simulationsframework (siehe Abbildung 5.2). . . . . 133
5.5. Kombiniertes Hard- und Software Smart Grid Simulationsmodell - aus [PBDR14a, S. 7]. . . . . 134
5.6. Konzeptionelle Architektur für den Simulation als ein Service Ansatz - aus [PDR15c, S. 4]. . . . . 139
5.7. UML Sequenzdiagramm das die hierarchische Komposition und Ausführung von Simulationsdiensten zeigt - nach [PDR15c]. . . . 140
5.8. UML Sequenzdiagramm das die parallele Komposition und Ausführung von Simulationsdiensten zeigt - aus [PDR15c, S. 5]. . 141
5.9. Konfigurationsbeispiele für drei exemplarische Implementierungsalternativen für den Simulation als ein Service Ansatz - nach [PDR15c]. . . . . 144
5.10. Ergebnisse der Skalierbarkeitsanalyse - nach [PDR15c]. . . . . 145
5.11. Daten-adaptiver *Simulation as a Service* Ansatz - aus [PDR15a, S. 11]. . . . . 146
5.12. Daten-adaptive Simulationsservice Architektur - nach [PDR15a]. 147

6.1. Verhalten einer Aktiven Komponente, das sich aus mehreren *Component Steps* zusammensetzt - aus [PDR15b, S. 3]. . . . . 153
6.2. Schichtenarchitektur der dezentralen Koordinationsmiddleware für Aktive Komponenten - nach [PDR15b, S. 4]. . . . . 156
6.3. Detailansicht einer Aktiven Komponente und ihrer Koordinationserweiterung - nach [PDR15b, S. 4]. . . . . 157
6.4. Selbstorganisierende Dynamik eines verteilten Systems basierend auf dezentralen Koordinationsprozessen - nach [PDR16a]. . . . . 162
6.5. Architektur des dezentralen Koordinationsframework - nach [PDR16a]. . . . . 163
6.6. UML-Klassendiagramm zur Schnittstelle für koordinierbare Komponenten. . . . . 165
6.7. UML Klassendiagramm des dezentralen Koordinationsframeworks mit den wichtigsten Klassen bzgl. der Koordination - nach [PDR16a]. . . . . 167

6.8. Visuelle Repräsentation eines generischen Koordinationsprozesses - aus [PDR16a, S. 7]. . . . . 168

6.9. UML Klassendiagramm der DeCoF Koordinationsbeschreibungssprache. . . . . 169

6.10. Visuelle Repräsentation der Koordinationsprozesse des Jade Party Beispiels - aus [PDR16a, S. 10]. . . . . 171

6.11. Zusammenhang zwischen selbstorganisierender Dynamik und struktureller Adaption im Konzept des dezentralen Koordinationsrahmenwerks. Die Abbildung unterscheidet sich von Abbildung 4.9 durch die Komponenten, die zur Umsetzung der strukturellen Adaptionsprozesse instruiert werden. . . . . 175

6.12. UML Klassendiagramm zur Schnittstelle für adaptierbare und koordinierbare Komponenten. . . . . 177

6.13. Architektur des dezentralen Koordinationsframeworks mit Unterstützung für strukturelle Adaptionen - nach [PDR16c]. Die Elemente in der Mitte, die im helleren Farbton abgebildet sind, stellen die Erweiterungen zur Unterstützung von strukturellen Adaptionen dar. Die Elemente am Rande, die im dunkleren Farbton abgebildet sind, zeigen die Elemente die bereits vorhandene Architektur des Frameworks (vgl. Abbildung 6.5). . . . . 178

6.14. UML Klassendiagramm des dezentralen Koordinationsframeworks mit den wichtigsten Klassen für die strukturellen Adaption. 180

6.15. UML Klassendiagramm der Erweiterung der DeCoF Koordinationsbeschreibungssprache für strukturelle Adaptionen. . . . . 181

7.1. Screenshot des MarsWorld Szenarios mit hervorgehobenen Erz-Clustern - aus [PRD15, S. 9]. . . . . 195

7.2. Screenshot der laufenden MarsWorld Anwendung mit hervorgehobenen, selbstorganisierend geformten lokalen Gruppen - aus [PRD15, S. 10]. . . . . 195

7.3. Evaluation der Koordinationsprozesse im MarsWorld Szenario aus [PRD15, S. 10]. Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse gibt Auskunft über den prozentualen Anteil an analysierten (*analyzed*), produzierten (*produced*) und eingesammelten (*collected*) Erz. . . . . 197

7.4. Effizienz der selbstorganisierenden Prozesse im MarsWorld Szenario aus [PRD15, S. 11]. Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse zeigt prozentual, wie viel effektiver die selbstorganisierenden Koordinationsprozesse gegenüber den anderen Koordinationsprozessen sind. . . . . 198

7.5. Auswertung der strukturellen Adaptionen im MarsWorld Szenario aus [PRD15, S. 11]. Die x-Achse zeigt die vergangene Zeit in $\frac{1}{10}$s Schritten. Die y-Achse zeigt die prozentuale Anzahl der gestarteten Abstimmungsprozesse (*Voting Attempts*), den prozentualen Anteil an simulierten Fehlern im Positionsservice (*Failures*), die prozentuale Anzahl an durchgeführten Adaptionen (*Adaptions*) und die relative Anzahl von Koordinationsnachrichten aus den Szenarien mit simulierten Fehler in Relation zur Anzahl von Koordinationsnachrichten aus einem Szenario ohne simulierten Fehlerfall (*rel. Messages*). . . . . 198
7.6. Auszug der simulierten Karte für das Bike-Sharing Beispiel, hier sind einige Radfahrer sowie die Ausleihstationen und der gewählte Kommunikationsradius gezeigt - aus [PDR16b, S. 7]. . . 205
7.7. Selbstorganisierende Dynamik des Fahrradverleihsystems - nach [PDR16b]. . . . . 205
7.8. Vergleich der Resultate für ein Simulationsszenario mit selbstorganisierendem Verhalten und einem ohne - aus [PDR16b, S. 8]. . 206
7.9. Mittlere Abweichung der Anzahl der normalen Stationen im Vergleich zu einem Referenzszenario ohne selbstorganisierendes Verhalten - aus [PDR16b, S. 8]. . . . . 207
7.10. Vergleich der Auswirkung der selbstorganisierenden und LKW-basierten Umverteilungsmaßnahmen im Bike-Sharing Beispiel - aus [PDR16c, S. 4] . . . . . 211
7.11. Auswirkung und Nachweis der Strukturelle Adaption der Koordinationsprozesse im Bike-Sharing Beispiel - nach [PDR16c, S. 4]. . . . . 213

8.1. Konzeptionelle Architektur und Entwicklungsansatz des Policen-basierten Ansatzes zur Einhaltung von nichtfunktionalen Anforderungen in selbstadaptiven und selbstorganisierenden Systemen - aus [Pre13, S. 4]. . . . . 235

# Tabellenverzeichnis

2.1. Klassifikation von Koordinationsmechanismen nach [Bra07]. . . 33
2.2. Problemorientiertes und musterbasiertes Verständnis von Koordinationsmechanismen - nach [WH06]. . . . . 36
2.3. Wesentliche Charakteristika selbstorganisierender Systeme - basierend auf [Vil13, S. 34] und erweitert um zusätzliche Referenzen. . . . . 37

7.1. Konfiguration der an der MarsWorld teilnehmenden Agenten. . . 194
7.2. Vergleich unterschiedlicher Ansätze zur Realisierung von dezentraler Koordination in verteilten Systemen. . . . . 218

# Quelltextverzeichnis

6.1. Deklarative XML Beschreibung der drei Koordinationsprozesse aus dem *Jade* Party Beispiel. . . . . 172

7.1. Deklarative Beschreibung der *latest_target_seen_nearest* Koordinationsprozess-Realisierung in der *MASDynamics* Sprache. 191
7.2. XML Quellcode der deklarativen Beschreibung der strukturellen Adaption für das MarsWorld Beispiel. . . . . 193
7.3. XML-basierte Beschreibung des Koordinationsprozesses zur Verbreitung der Auslastungsraten im Bike-Sharing Beispiel. . . . . 204
7.4. XML-basierte Beschreibung des Koordinationsprozesses zur Information des LKW-Operators über die aktuellen Auslastungsraten durch die Fahrradstationen im Bike-Sharing Beispiel. . . . . 210
7.5. XML-basierte Beschreibung der strukturellen Adaption im Bike-Sharing Beispiel. . . . . 212

A.1. XML Schema Definition (XSD) welche die Adaptionserweiterung der *MASDynamics* [SR09b] Sprache zeigt. . . . . 237
A.2. XML Schema Definition (XSD) zur Beschreibung von Koordinationsprozessen in einem verteilen System basierend auf Aktiven Komponenten. . . . . 238

# Literaturverzeichnis


[ABPA+09] ABOUZEID, AZZA, KAMIL BAJDA-PAWLIKOWSKI, DANIEL ABADI, AVI SILBERSCHATZ und ALEXANDER RASIN: *HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads*. Proc. VLDB Endow., 2(1):922–933, August 2009.

[ALMW09] ANDERSSON, JESPER, ROGÉRIO LEMOS, SAM MALEK und DANNY WEYNS: *Software Engineering for Self-Adaptive Systems*. Kapitel Modeling Dimensions of Self-Adaptive Software Systems, Seiten 27–47. Springer-Verlag, Berlin, Heidelberg, 2009.

[And02] ANDERSON, C.: *Self-organization in relation to several similar concepts: are the boundaries to self-organization indistinct?* The Biological bulletin, 202(3):247–255, 2002.

[Asc02] ASCHKE, MANFRED: *Kommunikation, Koordination und soziales System: theoretische Grundlagen für die Erklärung der Evolution von Kultur und Gesellschaft*. Lucius und Lucius, 2002.

[Ash47] ASHBY, W. R.: *Principles of the self-organizing dynamic system*. Journal of General Psychology, 37:125–128, 1947.

[AW09] AARTS, EMILE und REINER WICHERT: *Ambient intelligence*. In: *Technology Guide*, Seiten 244–249. Springer, 2009.

[Bal11] BALZERT, HELMUT: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Springer Spektrum, 3 Auflage, 2011.

[BBG06] BEYDEDA, S., M. BOCK und V. GRUHN: *Model-Driven Software Development*. Springer Berlin Heidelberg, 2006.

[BC06] BRUECKNER, S. und H. CZAP: *Organization, self-organization, autonomy and emergence: Status and challenges*. International Transactions on Systems Science and Applications, 2(1):1–9, 2006.

[BCB+14] BITTERMAN, THOMAS, PRASAD CALYAM, ALEX BERRYMAN, DAVID E. HUDAK, LIN LI, ALAN CHALKER, STEVE GORDON, DA ZHANG, DA CAI, CHANGPIL LEE und RAJIV RAMNATH: *Simulation as a service (SMaaS): a cloud-based framework to support the educational use of scientific software*. IJCC, 3(2):177–190, 2014.

[BCHM06] BERNON, C., V. CHEVRIER, V. HILAIRE und P. MARROW: *Applications of self-organising multi-agent systems: an initial framework of comparison*. Informatica, 30(1), 2006.

[BdBV08] BOER, C. A., A. DE BRUIN und A. VERBRAECK: *Distributed simulation in industry - a survey Part 3 - the HLA standard in industry*. In: *2008 Winter Simulation Conference*, Seiten 1094–1102, Dec 2008.

[BF01] BECK, KENT und MARTIN FOWLER: *Planning extreme programming*. Addison-Wesley Professional, 2001.

[BGLZ05] BRAVETTI, MARIO, ROBERTO GORRIERI, ROBERTO LUCCHI und GIANLUIGI ZAVATTARO: *Quantitative information in the tuple space coordination model*. Theoretical Computer Science, 346(1):28 – 57, 2005.

[BH06] BISHOP, J. und N. HORSPOOL: *Cross-Platform Development: Software that Lasts*. Computer, 39(10):26–35, Oct 2006.

[BHW07] BORDINI, R. H., J. F. HUBNER und M. WOOLDRIDGE: *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, 2007.

[BJM+05] BABAOGLU, OZALP, MÁRK JELASITY, ALBERTO MONTRESOR, CHRISTOF FETZER, STEFANO LEONARDI, AAD VAN MOORSEL und MAARTEN VAN STEEN: *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[BJP14] BRAUBACH, L., K. JANDER und A. POKAHR: *A Middleware for Managing Non-Functional Requirements in Cloud PaaS*. In: *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on*, Seiten 83–92, Sept 2014.

[BMM02] BABAOGLU, O., H. MELING und A. MONTRESOR: *Anthill: a framework for the development of agent-based peer-to-peer systems*. In: *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on*, Seiten 15–22, 2002.

[BMMSP06] BRANKE, J., M. MNIF, C. MULLER-SCHLOER und H. PROTHMANN: *Organic Computing - Addressing Complexity by Controlled Self-Organization*. In: *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on*, Seiten 185–191, Nov 2006.

[BMSG+09] BRUN, YURIY, GIOVANNA MARZO SERUGENDO, CRISTINA GACEK, HOLGER GIESE, HOLGER KIENLE, MARIN LITOIU,

HAUSI MÜLLER, MAURO PEZZÈ und MARY SHAW: *Software Engineering for Self-Adaptive Systems*. Kapitel Engineering Self-Adaptive Systems Through Feedback Loops, Seiten 48–70. Springer-Verlag, Berlin, Heidelberg, 2009.

[BMV05] BANDINI, STEFANIA, SARA MANZONI und GIUSEPPE VIZZARI: *Environments for Multi-Agent Systems: First International Workshop, E4MAS 2004, New York, NY, July 19, 2004, Revised Selected Papers*, Kapitel A Spatially Dependent Communication Model for Ubiquitous Systems, Seiten 74–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[BP12a] BRAUBACH, LARS und ALEXANDER POKAHR: *Developing Distributed Systems with Active Components and Jadex*. Scalable Computing: Practice and Experience, 13(2):100–119, 2012.

[BP12b] BRAUBACH, LARS und ALEXANDER POKAHR: *Intelligent Distributed Computing V: Proceedings of the 5th International Symposium on Intelligent Distributed Computing – IDC 2011, Delft, The Netherlands – October 2011*, Kapitel Addressing Challenges of Distributed Systems Using Active Components, Seiten 141–151. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BP12c] BRAUBACH, LARS und ALEXANDER POKAHR: *Jadex Active Components Framework - BDI Agents for Disaster Rescue Coordination*. In: *Software Agents, Agent Systems and Their Applications*, Seiten 57–84. 2012.

[BPG+04] BRESCIANI, PAOLO, ANNA PERINI, PAOLO GIORGINI, FAUSTO GIUNCHIGLIA und JOHN MYLOPOULOS: *Tropos: An Agent-Oriented Software Development Methodology*. Autonomous Agents and Multi-Agent Systems, 8(3):203–236, Mai 2004.

[BPJ11] BRAUBACH, LARS, ALEXANDER POKAHR und KAI JANDER: *Jadexcloud - an Infrastructure for Enterprise Cloud Applications*. In: *Proceedings of the 9th German Conference on Multi-agent System Technologies*, MATES'11, Seiten 3–15, Berlin, Heidelberg, 2011. Springer-Verlag.

[BPR99] BELLIFEMINE, FABIO, AGOSTINO POGGI und GIOVANNI RIMASSA: *JADE–A FIPA-compliant agent framework*. In: *Proceedings of PAAM*, Band 99, Seite 33. London, 1999.

[Bra87] BRATMAN, MICHAEL: *Intention, Plans, and Practical Reason*. Center for the Study of Language and Information, 1987.

[Bra07] BRAUBACH, LARS: *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*. Doktorarbeit, Fakultät für Mathematik, Informatik und Naturwissenschaften, Universität Hamburg, 2007.

[Brü00] BRÜCKNER, SVEN: *Return from the ant - synthetic ecosystems for manufacturing control*. Doktorarbeit, Mathematisch-Naturwissenschaftliche Fakultät II, Humboldt-Universität zu Berlin, 2000.

[BSP+02] BIGUS, J. P., D. A. SCHLOSNAGLE, J. R. PILGRIM, W. N. MILLS III und Y. DIAO: *ABLE: A toolkit for building multiagent autonomic systems*. IBM Systems Journal, 41(3):350–371, 2002.

[Büh08] BÜHRMANN, S.: *Bicycles as public-individual transport - European developments*. Technischer Bericht, Rupprecht Consult Forschung und Beratung GmbH, 2008.

[BVR+13] BRAUNAGEL, J., P. VUTHI, W. RENZ, H. SCHÄFERS, H. ZARIF und H. WIECHMANN: *Determination of load schedules and load shifting potentials of a high number of electrical consumers using mass simulation*. In: *Electricity Distribution (CIRED 2013), 22nd International Conference and Exhibition on*, Seiten 1–4, June 2013.

[Cam06] CAMAZINE, SCOTT: *Self-Organizing Systems*. John Wiley & Sons, Ltd, 2006.

[CAR+09] CRAMER, B., H. ANDRULEIT, H. REMPEL, H. BABIES, S SCHLÖMER, S. SCHMIDT, U. SCHWARZ.SCHAMOERA, N. OCHMANN, J. MESSNER und S. REHDER: *Energierohstoffe 2009: Reserven, Ressourcen, Verfügbarkeit*. Erdöl, Erdgas, Kohle, Kernbrennstoffe, Geothermische Energie. Bundesanstalt für Geowissenschaften und Rohstoffe (BGR), 2009.

[Cay13] CAYIRCI, E.: *Modeling and simulation as a cloud service: A survey*. In: *2013 Winter Simulations Conference (WSC)*, Seiten 389–400, Dec 2013.

[CCC+05] CHANG, PAUL HSUEH-MIN, KUANG-TAI CHEN, YU-HUNG CHIEN, EDWARD KAO und VON-WUN SOO: *Environments for Multi-Agent Systems: First International Workshop, E4MAS 2004, New York, NY, July 19, 2004, Revised Selected Papers*, Kapitel From Reality to Mind: A Cognitive Middle Layer of Environment Concepts for Believable Agents, Seiten 57–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[CDnJS+03] CAMAZINE, SCOTT, JEAN-LOUIS DENEUBOURG, NIGEL R. FRANKS NAD JAMES SNEYD, GUY THERAULAZ und ERIC BONABEAU: *Self-Organization in Biological Systems*. Princeton Studies in Complexity. Princeton University Press, 2003.

[CGD13] COOLS, SEUNG-BAE, CARLOS GERSHENSON und BART D'HOOGHE: *Advances in Applied Self-Organizing Systems*,

Kapitel Self-Organizing Traffic Lights: A Realistic Simulation, Seiten 45–55. Springer London, London, 2013.

[CGS05] CHENG, SHANG-WEN, DAVID GARLAN und BRADLEY SCHMERL: *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, Kapitel Making Self-Adaptation an Engineering Reality, Seiten 158–173. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[CK09] CANZLER, W. und A. KNIE: *Grüne Wege aus der Autokrise. Vom Autobauer zum Mobilitätsdienstleister. Ein Strategiepapier.* Technischer Bericht, Heinrich-Böll-Stiftung, 2009.

[CLG+09] CHENG, BETTY H. C., ROGÉRIO LEMOS, HOLGER GIESE, PAOLA INVERARDI, JEFF MAGEE, JESPER ANDERSSON, BASIL BECKER, NELLY BENCOMO, YURIY BRUN, BOJAN CUKIC, GIOVANNA MARZO SERUGENDO, SCHAHRAM DUSTDAR, ANTHONY FINKELSTEIN, CRISTINA GACEK, KURT GEIHS, VINCENZO GRASSI, GABOR KARSAI, HOLGER M. KIENLE, JEFF KRAMER, MARIN LITOIU, SAM MALEK, RAFFAELA MIRANDOLA, HAUSI A. MÜLLER, SOOYONG PARK, MARY SHAW, MATTHIAS TICHY, MASSIMO TIVOLI, DANNY WEYNS und JON WHITTLE: *Software Engineering for Self-Adaptive Systems*, Kapitel Software Engineering for Self-Adaptive Systems: A Research Roadmap, Seiten 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[CMT04] CHARLES, AHMED, RONALDO MENEZES und ROBERT TOLKSDORF: *On the Implementation of SwarmLinda*. In: *Proceedings of the 42Nd Annual Southeast Regional Conference*, ACM-SE 42, Seiten 297–298, New York, NY, USA, 2004. ACM.

[CNYM99] CHUNG, LAWRENCE, BRIAN NIXON, ERIC YU und JOHN MYLOPOULOS: *Non-functional requirements in Software Engineering*. Kluwer Academic Publishers, 1999.

[Coh09] COHN, MIKE: *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional, 2009.

[Cor91] CORKILL, DANIEL D.: *Blackboard Systems*. AI Expert, 6(9):40–47, 1991.

[CSG08] CHASSIN, D. P., K. SCHNEIDER und C. GERKENSMEYER: *GridLAB-D: An open-source power systems modeling and simulation environment*. In: *2008 IEEE/PES Transmission and Distribution Conference and Exposition*, Seiten 1–5, April 2008.

[CYN14] CHAVALI, P., P. YANG und A. NEHORAI: *A Distributed Algorithm of Appliance Scheduling for Home Energy Management*

*System*. IEEE Transactions on Smart Grid, 5(1):282–290, Jan 2014.

[Das08] DASTANI, MEHDI: *2APL: a practical agent programming language*. Autonomous Agents and Multi-Agent Systems, 16(3):214–248, 2008.

[Dav01] DAVIDSSON, PAUL: *Multi Agent Based Simulation: Beyond Social Simulation*, Seiten 97–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[DB06] DORIGO, M. und M. BIRATTARI: *Ant Colony Optimization*. IEEE Computational Intelligence Magazine, (1):28–39, 2006.

[DBP+15] DETHLEFS, TIM, CHRISTOPH BRUNNER, THOMAS PREISLER, OLIVER RENKE, WOLFGANG RENZ und ANDREA SCHRÖDER: *Energy Service Description for Capabilities of Distributed Energy Resources*. In: GOTTWALT, SEBASTIAN, LUKAS KÖNIG und HARTMUT SCHMECK (Herausgeber): *Energy Informatics - 4th D-A-CH Conference, EI 2015, Karlsruhe, Germany, November 12-13, 2015, Proceedings*, Band 9424 der Reihe *Lecture Notes in Computer Science*, Seiten 24–35. Springer, 2015.

[DBPR14] DETHLEFS, TIM, DIRK BEEWEN, THOMAS PREISLER und WOLFGANG RENZ: *A consumer-orientated Architecture for distributed Demand-Side-Optimization*. In: GÓMEZ, JORGE MARX et al. [GSV+14], Seiten 349–356.

[DBU11] DERKSEN, C., C. BRANKI und R. UNLAND: *Agent. GUI: A multi-agent based simulation framework*. In: *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, Seiten 623–630, Sept 2011.

[Dey01] DEY, ANIND K.: *Understanding and Using Context*. Personal Ubiquitous Comput., 5(1):4–7, Januar 2001.

[DG04] DEMAIO, P. und J. GIFFORD: *Will Smart Bikes Succeed as Public Transportation in the United States?* Journal of Public Transportation, 7(2):1–15, 2004.

[dLF02] LEMOS, ROGÉRIO DE und JOSÉ LUIZ FIADEIRO: *An Architectural Support for Self-adaptive Software for Treating Faults*. In: *Proceedings of the First Workshop on Self-healing Systems*, WOSS '02, Seiten 39–42, New York, NY, USA, 2002. ACM.

[DMSFR10] DI MARZO SERUGENDO, GIOVANNA, JOHN FITZGERALD und ALEXANDER ROMANOVSKY: *MetaSelf: An Architecture and a Development Method for Dependable Self-* Systems*. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, Seiten 457–461, New York, NY, USA, 2010. ACM.

[DMSGK05] DI MARZO SERUGENDO, GIOVANNA, MARIE-PIERRE GLEIZES und ANTHONY KARAGEORGOS: *Self-organization in Multi-agent Systems*. Knowl. Eng. Rev., 20(2):165–189, Juni 2005.

[DMSGK11] DI MARZO SERUGENDO, GIOVANNA, MARIE-PIERRE GLEIZES und ANTHONY KARAGEORGOS (Herausgeber): *Self-Organising Software*. Natural Computing Series. Springer-Verlag Berlin Heidelberg, 1 Auflage, 2011.

[DPR14] DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *Multi-Agent-based Distributed Optimization for Demand-Side-Management Applications*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems, Warsaw, Poland, September 7-10, 2014.*, Seiten 1489–1496, 2014.

[DPR15a] DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *Ant-Colony based Self-Optimization for Demand-Side-Management*. In: *SmartER Europe 2015, Essen, Germany*, 2015.

[DPR15b] DETHLEFS, TIM, THOMAS PREISLER und WOLFGANG RENZ: *A DER Registry System as an Infrastructural Component for future Smart Grid Applications*. In: *ETG Congress 2015 - Die Energiewende, Bonn, Deutschland, 17.-18. November*, 2015.

[DTc+04] DORIGO, MARCO, VITO TRIANNI, EROL ŞAHIN, RODERICH GROSS, THOMAS H. LABELLA, GIANLUCA BALDASSARRE, STEFANO NOLFI, JEAN-LOUIS DENEUBOURG, FRANCESCO MONDADA, DARIO FLOREANO und LUCA M. GAMBARDELLA: *Evolving Self-Organizing Behaviors for a Swarm-Bot*. Auton. Robots, 17(2-3):223–245, September 2004.

[DWH05] DE WOLF, TOM und TOM HOLVOET: *Towards a Methodology for Engineering Self-Organising Emergent Systems*. In: *Proceedings of the 2005 Conference on Self-Organization and Autonomic Informatics (I)*, Seiten 18–34, Amsterdam, The Netherlands, The Netherlands, 2005. IOS Press.

[DWSH05] DE WOLF, TOM, GIOVANNI SAMAEY und TOM HOLVOET: *Engineering self-organising emergent systems with simulation-based scientific analysis*. In: *Proceedings of the Fourth International Workshop on Engineering Self-Organising Applications*, Seiten 146–160, 2005.

[Edm05a] EDMONDS, BRUCE: *Engineering Self-Organising Systems: Methodologies and Applications*, Kapitel Using the Experimen-

tal Method to Produce Reliable Self-Organised Systems, Seiten 84–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Edm05b] EDMONDS, BRUCE: *Using the Experimental Method to Produce Reliable Self-Organised Systems*, Seiten 84–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Erl04] ERL, T.: *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall PTR, 2004.

[ESW07] EISENBACH, SUSAN, CHRIS SADLER und DOMINIC WONG: *Component Adaptation in Contemporary Execution Environments*. In: *Proceedings of the 7th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, DAIS'07, Seiten 90–103, Berlin, Heidelberg, 2007. Springer-Verlag.

[FE98] FRITZSON, PETER und VADIM ENGELSON: *Modelica — A unified object-oriented language for system modeling and simulation*, Seiten 67–90. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[Fer99] FERBER, JACQUES: *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st Auflage, 1999.

[FG97] FRANKLIN, STAN und ART GRAESSER: *Intelligent Agents III Agent Theories, Architectures, and Languages: ECAI'96 Workshop (ATAL) Budapest, Hungary, August 12–13, 1996 Proceedings*, Kapitel Is It an agent, or just a program?: A taxonomy for autonomous agents, Seiten 21–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[FGM04] FERBER, JACQUES, OLIVIER GUTKNECHT und FABIEN MICHEL: *Agent-Oriented Software Engineering IV: 4th InternationalWorkshop, AOSE 2003, Melbourne, Australia, July 15, 2003. Revised Papers*, Kapitel From Agents to Organizations: An Organizational View of Multi-agent Systems, Seiten 214–230. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Fie00] FIELDING, ROY THOMAS: *Architectural styles and the design of network-based software architectures*. Doktorarbeit, University of California, Irvine, 2000.

[FNO09] FROEHLICH, JON, JOACHIM NEUMANN und NURIA OLIVER: *Sensing and Predicting the Pulse of the City Through Shared Bicycling*. In: *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, IJCAI'09, Seiten 1420–1426, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[GB82] GELERNTER, DAVID und ARTHUR J. BERNSTEIN: *Distributed Communication via Global Buffer*. In: *Proceedings of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*, PODC '82, Seiten 10–18, New York, NY, USA, 1982. ACM.

[GBH11] GUO, S., F. BAI und X. HU: *Simulation software as a service and Service-Oriented simulation experiment*. In: *Information Reuse and Integration (IRI), 2011 IEEE International Conference on*, Seiten 113–116, Aug 2011.

[GC92] GELERNTER, DAVID und NICHOLAS CARRIERO: *Coordination Languages and Their Significance*. Commun. ACM, 35(2):97–107, Februar 1992.

[GC03] GANEK, A. G. und T. A. CORBI: *The Dawning of the Autonomic Computing Era*. IBM Systems Journal, 42(1):5–18, Januar 2003.

[GCH+04] GARLAN, D., S. W. CHENG, A. C. HUANG, B. SCHMERL und P. STEENKISTE: *Rainbow: architecture-based self-adaptation with reusable infrastructure*. Computer, 37(10):46–54, Oct 2004.

[Gei08] GEIHS, KURT: *Selbst-adaptive Software*. Informatik-Spektrum, 31:133–145, 2008.

[Gel85] GELLINGS, C. W.: *The concept of demand-side management for electric utilities*. Proceedings of the IEEE, 73(10):1468–1470, Oct 1985.

[Ger06] GERSHENSON, CARLOS: *A General Methodology for Designing Self-Organizing Systems*. http://arxiv.org/abs/nlin/0505009, Februar 2006.

[Ger07] GERSHENSON, CARLOS: *Design and Control of Self-Organizing Systems*. Doktorarbeit, Vrije Universiteit Brussel, 2007.

[GG09] GALUBA, WOJCIECH und SARUNAS GIRDZIJAUSKAS: *Distributed hash table*. In: *Encyclopedia of Database Systems*, Seiten 903–904. Springer, 2009.

[GKL06] GEIMER, M., T. KRÜGER und P. LINSEL: *Co-Simulation, gekoppelte Simulation oder Simulatorkopplung? Ein Versuch der Begriffsvereinheitlichung*. O+P Ölhydraulik und Pneumatik, 50(11-12):572–576, 2006.

[GLS99] GROPP, W., E. LUSK und A. SKJELLUM: *Using MPI: portable parallel programming with the message-passing interface*. MIT Press, 1999.

[GM05] GOUAICH, ABDELKADER und FABIEN MICHEL: *Towards a Unified View of the Environment(s) within Multi-Agent Systems*. Informatica, 29(4):423–432, 2005.

[GMR+03] GABALDON, A., A. MOLINA, C. ROLDAN, J. A. FUENTES, E. GOMEZ, I. J. RAMIREZ-ROSADO, P. LARA, J. A. DOMINGUEZ, E. GARCIA-GARRIDO und E. TARANCON: *Assessment and simulation of demand-side management potential in urban power distribution networks*. In: *Power Tech Conference Proceedings, 2003 IEEE Bologna*, Band 4, Seite 5, jun 2003.

[GNO+08] GÜDEMANN, M., F. NAFZ, F. ORTMEIER, H. SEEBACH und W. REIF: *A Specification and Construction Paradigm for Organic Computing Systems*. In: *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, Seiten 233–242, Oct 2008.

[Gra59] GRASSE, PIERRE-PAUL: *La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs*. Insectes Sociaux, 4:41–81, 1959.

[GS12] GUINEA, S. und P. SAEEDI: *Coordination of distributed systems through self-organizing group topologies*. In: *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, Seiten 63–72, June 2012.

[GSV+14] GÓMEZ, JORGE MARX, MICHAEL SONNENSCHEIN, UTE VOGEL, ANDREAS WINTER, BARBARA RAPP und NILS GIESEN (Herausgeber): *28th International Conference on Informatics for Environmental Protection: ICT for Energy Effieciency, EnviroInfo 2014, Oldenburg, Germany, September 10-12, 2014*. BIS-Verlag, 2014.

[Hak06] HAKEN, HERMANN: *Information and Self-Organization: A Macroscopic Approach to Complex Systems*. Springer-Verlag Berlin Heidelberg, 3 Auflage, 2006.

[HATC+16] HÜNING, CHRISTIAN, MITJA ADEBAHR, THOMAS THIEL-CLEMEN, JAN DALSKI, ULFIA LENFERS und LUKAS GRUNDMANN: *Modeling and Simulation As a Service with the Massive Multi-agent System MARS*. In: *Proceedings of the Agent-Directed Simulation Symposium*, ADS '16, Seiten 1:1–1:8, San Diego, CA, USA, 2016. Society for Computer Simulation International.

[Her08] HERRMANN, KLAUS: *Self-organizing ambient intelligence: principles, algorithms, and protocols*. Doktorarbeit, Technische Universität Berlin, 2008.

[Hey99] HEYLIGHEN, FRANCIS: *The Science Of Self-Organization And Adaptivity*. In: *in: Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS*, Seiten 253–280. Publishers Co. Ltd, 1999.

[HGRZ06] HERDEN, SEBASTIAN, JORGE MARX GOMEZ, CLAUS RAUTENSTRAUCH und ANDRE ZWANZIGER: *Software-Architekturen für das E-Business*. eXamen.press. Springer-Verlag Berlin Heidelberg, 1 Auflage, 2006.

[HKZ+11] HINDMAN, BENJAMIN, ANDY KONWINSKI, MATEI ZAHARIA, ALI GHODSI, ANTHONY D. JOSEPH, RANDY KATZ, SCOTT SHENKER und ION STOICA: *Mesos: A Platform for Fine-grained Resource Sharing in the Data Center*. In: *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, Seiten 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[HL11] HILBERT, MARTIN und PRISCILA LÓPEZ: *The World's Technological Capacity to Store, Communicate, and Compute Information*. Science, 332(6025):60–65, 2011.

[HM08] HUEBSCHER, MARKUS C. und JULIE A. MCCANN: *A Survey of Autonomic Computing&Mdash;Degrees, Models, and Applications*. ACM Comput. Surv., 40(3):7:1–7:28, August 2008.

[HNS+11] HOFFMANN, ALWIN, FLORIAN NAFZ, ANDREAS SCHIERL, HELLA SEEBACH und WOLFGANG REIF: *Developing Self-Organizing Robotic Cells Using Organic Computing Principles*, Seiten 253–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Hol93] HOLLAND, JOHN: *The effect of labels (tags) on social interactions*. Technischer Bericht, Santa Fe Institute Working Paper 93-10-064. Santa Fe, NM, 1993.

[Hol99] HOLLAND, JOHN H.: *Emergence: From Chaos to Order*. Perseus Publishing, 1999.

[HS99] HUHNS, MICHAEL N. und LARRY M. STEPHENS: *Multiagent Systems*. Kapitel Multiagent Systems and Societies of Agents, Seiten 79–120. MIT Press, Cambridge, MA, USA, 1999.

[HS06] HINCHEY, M. G. und R. STERRITT: *Self-managing software*. Computer, 39(2):107–109, Feb 2006.

[HSB07] HUBNER, JOMI F., JAIME S. SICHMAN und OLIVIER BOISSIER: *Developing Organised Multiagent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels*. Int. J. Agent-Oriented Softw. Eng., 1(3/4):370–395, Dezember 2007.

[HW94] HÖLLDOBLER, B. und E. O. WILSON: *Journey to the Ants*. Bellknap Press/Harvard University Press, 1994.

[HW08] HAKLAY, M. und P. WEBER: *OpenStreetMap: User-Generated Street Maps*. Pervasive Computing, IEEE, 7(4):12–18, Oct 2008.

[HWFTC14] HÜNING, CHRISTIAN, JASON WILMANS, NILS FEYERABEND und THOMAS THIEL-CLEMEN: *MARS - A next-gen multi-agent simulation framework*. In: WITTMANN, J. und D. MARETIS (Herausgeber): *Simulation in Umwelt- und Geowissenschaften, Workshop Osnabrück*. GI, Shaker, 2014.

[HWM06] HERRMANN, KLAUS, MATTHIAS WERNER und GERO MÜHL: *A Methodology for Classifying Self-Organizing Software Systems*. International Transactions on Systems Science and Applications, 2(1):41–50, 2006.

[HWMH06] HERRMANN, KLAUS, MATTHIAS WERNER, GERO MÜHL und HANS-ULRICH HEISS: *Ein methodischer Ansatz zur Klassifizierung selbstorganisierender Softwaresysteme*. In: *Selbstorganisierende, Adaptive, Kontextsensitive verteilte Systeme (SAKS)*, Fachgespräch der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme, Kassel, Germany, mar 2006. (in German).

[IEE00] *Ieee standard for modeling and simulation (m&s) high level architecture - framework and rules*, 2000.

[ISS99] ISERMANN, R., J. SCHAFFNIT und S. SINSEL: *Hardware-in-the-loop simulation for the design and testing of engine-control systems*. Control Engineering Practice, 7(5):643 – 653, 1999.

[JB03] JENNINGS, NICHOLAS R. und STEFAN BUSSMANN: *Agent-Based Control Systems*. IEEE Control Systems Magazine, 23:61–74, 2003.

[JBP10] JANDER, KAI, LARS BRAUBACH und ALEXANDER POKAHR: *EnvSupport: A Framework for Developing Virtual Environments*. In: TRAPPL, ROBERT (Herausgeber): *In Proceedings of International Workshop From Agent Theory to Agent Implementation (AT2AI-7)*, Seiten 471–476. Austrian Society for Cybernetic Studies Vienna, Austria, 2010.

[JBR99] JACOBSON, IVAR, GRADY BOOCH und JAMES RUMBAUGH: *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[Joh13] JOHNS, MAT: *Getting Started with Hazelcast*. Packt Publishing Ltd, 2013.

[JR02] JULIEN, CHRISTINE und GRUIA-CATALIN ROMAN: *Egocentric Context-aware Programming in Ad Hoc Mobile Environments*. In: *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, SIGSOFT '02/FSE-10, Seiten 21–30, New York, NY, USA, 2002. ACM.

[JXJY06] JIANG, YI, GUANGTAO XUE, ZHAOQING JIA und JINYUAN YOU: *DTuples: A Distributed Hash Table Based Tuple Space Service for Distributed Coordination*. In: *Proceedings of the Fifth International Conference on Grid and Cooperative Computing*, GCC '06, Seiten 101–106, Washington, DC, USA, 2006. IEEE Computer Society.

[Kam10] KAMPER, ANDREAS: *Dezentrales Lastmanagement zum Ausgleich kurzfristiger Abweichungen im Stromnetz*. KIT Scientific Publishing, 2010.

[Kar08] KARSENTI, ERIC: *Self-organization in cell biology: a brief history*. Nature Reviews Molecular Cell Biology, 9:255–262, 2008.

[Kar11] KARNOUSKOS, S.: *Cyber-Physical Systems in the SmartGrid*. In: *2011 9th IEEE International Conference on Industrial Informatics*, Seiten 20–23, July 2011.

[KC03] KEPHART, J.O. und D.M. CHESS: *The vision of autonomic computing*. Computer, 36(1):41–50, Jan 2003.

[KG06] KEIL, DAVID und DINA GOLDIN: *Indirect Interaction in Environments for Multi-agent Systems*. In: *Proceedings of the 2Nd International Conference on Environments for Multi-Agent Systems*, E4MAS'05, Seiten 68–87, Berlin, Heidelberg, 2006. Springer-Verlag.

[KGGP09] KADDOUM, E., M. P. GLEIZES, J. P. GEORGE und G. PICARD: *Characterizing and Evaluating Problem Solving Self-* Systems*. In: *Computation World*, Seiten 137–145, Nov 2009.

[KH01] KICZALES, GREGOR und ERIK HILSDALE: *Aspect-oriented Programming*. SIGSOFT Softw. Eng. Notes, 26(5):313–313, September 2001.

[Kim95] KIM, J.: *Problems in the Philosophy of Mind*. Oxford Companion to Philosophy. Oxford University Press, Oxford, 1995.

[Klü01] KLÜGL, FRANZISKA: *Multiagentensimulation.: Konzepte, Werkzeuge, Anwendungen*. Agententechnologie Series. Addison Wesley Verlag, 2001.

[KMG+10] KALTENBRUNNER, ANDREAS, RODRIGO MEZA, JENS GRIVOLLA, JOAN CODINA und RAFAEL BANCHS: *Urban Cycles and Mobility Patterns: Exploring and Predicting Trends in a Bicycle-based Public Transport System*. Pervasive and Mobile Computing, 6(4):455–466, August 2010.

[KPGV03] KAISER, G., J. PAREKH, P. GROSS und G. VALETTO: *Kinesthetics eXtreme: an external infrastructure for monitoring distributed legacy systems*. In: *Autonomic Computing Workshop. 2003. Proceedings of the*, Seiten 22–30, June 2003.

[KWK05] KOK, J. K., C. J. WARMER und I. G. KAMPHUIS: *PowerMatcher: Multiagent Control in the Electricity Infrastructure*. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, Seiten 75–82, New York, NY, USA, 2005. ACM.

[LPK+14] LI, LIXIANG, HAIPENG PENG, JÜRGEN KURTHS, YIXIAN YANG und HANS JOACHIM SCHELLNHUBER: *Chaos-order transition in foraging behavior of ants*. Proceedings of the National Academy of Sciences, 111(23):8392–8397, 2014.

[LSS12] LOGENTHIRAN, T., D. SRINIVASAN und T. Z. SHUN: *Demand Side Management in Smart Grid Using Heuristic Optimization*. IEEE Transactions on Smart Grid, 3(3):1244–1252, Sept 2012.

[LY02] LYYTINEN, KALLE und YOUNGJIN YOO: *Ubiquitous computing*. Communications of the ACM, 45(12):63–96, 2002.

[Mar91] MARTIN, JAMES: *Rapid Application Development*. Macmillan Coll Div, 1991.

[Mar00] MARTIN, JAMES: *Application Development without Programmers*. Prentice Hall PTR, Upper Sadie River, NJ, USA, 2000.

[Mar10] MARTINEZ, M.: *Washington, D.C. launches the nation's largest bike shar-ing program*. Grist Magazin, 2010.

[Mau13] MAUS, JONATHAN: *Behind the scenes of Capital Bikeshare*. `http://bikeportland.org/2013/03/10/behind-the-scenes-of-capital-bikeshare-84006`, 2013. Aufgerufen am: 14.03.2016.

[MBAG06] MANO, JEAN-PIERRE, CHRISTINE BOURJOT, GABRIEL ALEJANDRO und PIERRE GLIZE: *Bio-inspired Mechanisms for Artificial Self-organised Systems*. Informatica (Slovenia), 30(1):55–62, 2006.

[MC94] MALONE, T. und K. CROWSTON: *The interdisciplinary study of coordination*. ACM Computing Surveys, 26(1):87–119, 1994.

[MCCG+12] MATALLANAS, E., M. CASTILLO-CAGIGAL, A. GUTIERREZ, F. MONASTERIO-HUELIN, E. CAAMANO-MARTIN, D. MASA und J. JIMENEZ-LEUBE: *Neural network controller for Active Demand-Side Management with {PV} energy in the residential sector*. Applied Energy, 91(1):90 – 97, 2012.

[MG09] MELL, PETER und TIM GRANCE: *The NIST Definition of Cloud Computing*. Technischer Bericht, National Institute of Standards and Technology, Information Technology Laboratory, 2009.

[MH04] MCCANN, JULIE A. und MARKUS C. HUEBSCHER: *Grid and Cooperative Computing - GCC 2004 Workshops*, Band 3252 der Reihe *Lecture Notes in Computer Science*, Kapitel Evaluation Issues in Autonomic Computing, Seiten 597–608. Springer Berlin Heidelberg, 2004.

[Mid09] MIDGLEY, P.: *The Role of Smart Bike-sharing Systems*. Urban Mobility Journeys, Seiten 23–31, 2009.

[MMG+09] MORANDINI, MIRKO, FRÉDÉRIC MIGEON, MARIE-PIERRE GLEIZES, CHRISTINE MAUREL, LORIS PENSERINI und ANNA PERINI: *A Goal-Oriented Approach for Modelling Self-organising MAS*. In: *Proceedings of the 10th International Workshop on Engineering Societies in the Agents World X*, ESAW '09, Seiten 33–48, Berlin, Heidelberg, 2009. Springer-Verlag.

[MMTZ06] MAMEI, MARCO, RONALDO MENEZES, ROBERT TOLKSDORF und FRANCO ZAMBONELLI: *Case studies for self-organization in computer science*. Journal of Systems Architecture, 52(8–9):443 – 460, 2006. Nature-Inspired Applications and Systems.

[Mog06] MOGUL, JEFFREY C.: *Emergent (Mis)Behavior vs. Complex Software Systems*. SIGOPS Oper. Syst. Rev., 40(4):293–304, April 2006.

[MPP08] MORANDINI, MIRKO, LORIS PENSERINI und ANNA PERINI: *Towards Goal-oriented Development of Self-adaptive Systems*. In: *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, SEAMS '08, Seiten 9–16, New York, NY, USA, 2008. ACM.

[MR09] MARINO, JIM und MICHAEL ROWLEY: *Understanding SCA (Service Component Architecture)*. Addison-Wesley Professional, 1st Auflage, 2009.

[MS04] MÜLLER-SCHLOER, C.: *Organic computing - on the feasibility of controlled emergence*. In: *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, Seiten 2–5, Sept 2004.

[MS09] MARZO SERUGENDO, GIOVANNA: *Stabilization, Safety, and Security of Distributed Systems: 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*, Kapitel Robustness and Dependability of Self-Organizing Systems - A Safety Engineering Perspective, Seiten 254–268. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[MSKC04a] MCKINLEY, PHILIP K., SEYED MASOUD SADJADI, ERIC P. KASTEN und BETTY H. C. CHENG: *Composing Adaptive Software*. Computer, 37(7):56–64, Juli 2004.

[MSKC04b] MCKINLEY, PHILIP. K., SEYED MASOUD SADJADI, ERIC P. KASTEN und BETTY H. C. CHENG: *A Taxonomy of Compositional Adaptation*. Technischer Bericht MSU-CSE-04-17, Department of Computer Science and Engineering, Michigan State University, 2004.

[MWJ+07] MUEHL, G., M. WERNER, M. A. JAEGER, K. HERRMANN und H. PARZYJEGLA: *On the Definitions of Self-Managing and Self-Organizing Systems*. In: *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*, Seiten 1–11, Feb 2007.

[MZ06a] MAMEI, MARCO und FRANCO ZAMBONELLI: *Field-Based Coordination for Pervasive Multiagent Systems*. Springer Series on Agent Technology. Springer-Verlag Berlin Heidelberg, 1 Auflage, 2006.

[MZ06b] MAMEI, MARCO und FRANCO ZAMBONELLI: *Self-maintained distributed tuples for field-based coordination in dynamic networks*. Concurrency and Computation: Practice and Experience, 18(4):427–443, 2006.

[MZL04] MAMEI, M., F. ZAMBONELLI und L. LEONARDI: *Cofields: a physically inspired approach to motion coordination*. IEEE Pervasive Computing, 3(2):52–61, April 2004.

[Nii86] NII, H PENNY: *The blackboard model of problem solving and the evolution of blackboard architectures*. AI magazine, 7(2):38, 1986.

[NLPT14] NICOLA, ROCCO DE, MICHELE LORETI, ROSARIO PUGLIESE und FRANCESCO TIEZZI: *A Formal Approach to Autonomic Systems Programming: The SCEL Language*. ACM Trans. Auton. Adapt. Syst., 9(2):7:1–7:29, Juli 2014.

[NVCO10] NARDINI, ELENA, MIRKO VIROLI, MATTEO CASADEI und ANDREA OMICINI: *A Self-Organising Infrastructure for Chemical-Semantic Coordination: Experiments in TuCSoN*. In: OMICINI, ANDREA und MIRKO VIROLI (Herausgeber): *Proceedings of the*

*11th WOA 2010 Workshop, Dagli Oggetti Agli Agenti, Rimini, Italy, September 5-7, 2010.*, Band 621 der Reihe *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[OAS07] *Web Services Business Process Execution Language Version 2.0*, 2007.

[Ode00] ODELL, PETER R.: *The global energy market in the long term: the continuing dominance of affordable non-renewable resources*. Energy, Exploration & Exploitation, 18(5):599–613, 2000.

[OMG11] *Business Process Model And Notation*, 2011.

[OPB03] ODELL, JAMES, H. VAN DYKE PARUNAK und BERNHARD BAUER: *Practical Foundations of Business System Specifications*, Kapitel Extending UML for Agent-Based Systems, Seiten 245–270. Springer Netherlands, Dordrecht, 2003.

[OPF03] ODELL, JAMES, H. VAN DYKE PARUNAK und MITCHELL FLEISCHER: *Modeling Agents and their Environment: The Communication Environment*. Journal of Object Technology, 2(3):39–52, May-June 2003.

[ORV+04] OMICINI, ANDREA, ALESSANDRO RICCI, MIRKO VIROLI, CRISTIANO CASTELFRANCHI und LUCA TUMMOLINI: *Coordination Artifacts: Environment-Based Coordination for Intelligent Agents*. In: *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, Seiten 286–293, Washington, DC, USA, 2004. IEEE Computer Society.

[PB05] PREHOFER, C. und C. BETTSTETTER: *Self-organization in communication networks: principles and design paradigms*. Communications Magazine, IEEE, 43(7):78–85, July 2005.

[PB10] POKAHR, ALEXANDER und LARS BRAUBACH: *Multi-agent Systems: Decentral approaches for designing, organizing, and operating information systems*, Kapitel The Notions of Application, Spaces and Agents - New Concepts for Constructing Agent Applications, Seiten 159 – 170. Universitätsverlag Göttingen, 2010.

[PB13] POKAHR, ALEXANDER und LARS BRAUBACH: *The Active Components Approach for Distributed Systems Development*. Int. J. Parallel Emerg. Distrib. Syst., 28(4):321–369, August 2013.

[PBDR14a] PREISLER, THOMAS, GREGOR BALTHASAR, TIM DETHLEFS und WOLFGANG RENZ: *Scalable Integration of 4GL-Models and Algorithms for massive Smart Grid Simulations and Applications*. In: GÓMEZ, JORGE MARX et al. [GSV+14], Seiten 341–348.

[PBDR14b] PREISLER, THOMAS, GREGOR BALTHASAR, TIM DETHLEFS und WOLFGANG RENZ: *Servicekomponenten-basierte Architektur für mikroskopische und makroskopische Simulation der städtischen Energieversorgung*. In: *VDE-Kongress 2014 Smart Cities, Frankfurt am Main, Deutschland*, 2014.

[PBJ10] POKAHR, ALEXANDER, LARS BRAUBACH und KAI JANDER: *Multiagent System Technologies: 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings*, Kapitel Unifying Agent and Component Concepts, Seiten 100–112. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[PBJ13] POKAHR, ALEXANDER, LARS BRAUBACH und KAI JANDER: *The Jadex Project: Programming Model*, Seiten 21–53. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[PBL05] POKAHR, ALEXANDER, LARS BRAUBACH und WINFRIED LAMERSDORF: *Agenten: Technologie für den Mainstream*. it - Information Technology, 47(5):300–307, September 2005.

[PDR15a] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Data-Adaptive Simulation: Cooperativeness of Users in Bike-Sharing Systems*. In: KERSTEN, WOLFGANG, THORSTEN BLECKER und CHRISTIAN M. RINGLE (Herausgeber): *Proceedings of the Hamburg International Conference of Logistics*, Band 20. epubli GmbH, 2015.

[PDR15b] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Middleware for Constructing Decentralized Control in Self-Organizing Systems*. In: *2015 IEEE International Conference on Autonomic Computing, Grenoble, France, July 7-10, 2015*, Seiten 325–330. IEEE, 2015.

[PDR15c] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Simulation as a service: A design approach for large-scale energy network simulations*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, Lódz, Poland, September 13-16, 2015*, Seiten 1765–1772. IEEE, 2015.

[PDR16a] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *DeCoF: A Decentralized Coordination Framework for Various Multi-Agent Systems*. In: KLUSCH, MATTHIAS, RAINER UNLAND, ONN SHEHORY, ALEXANDER POKAHR und SEBASTIAN AHRNDT (Herausgeber): *Multiagent System Technologies - 14th German Conference, MATES 2016, Klagenfurt, Österreich,*

*September 27-30, 2016. Proceedings*, Band 9872 der Reihe *Lecture Notes in Computer Science*, Seiten 73–88. Springer, 2016.

[PDR16b] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Self-Organizing Redistribution of Bicycles in a Bike-Sharing System based on Decentralized Control*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016.*, Seiten 1471–1480, 2016.

[PDR16c] PREISLER, THOMAS, TIM DETHLEFS und WOLFGANG RENZ: *Structural Adaptations of Decentralized Coordination Processes in Self-Organizing Systems*. In: KOUNEV, SAMUEL, HOLGER GIESE und JIE LIU (Herausgeber): *2016 IEEE International Conference on Autonomic Computing, ICAC 2016, Wuerzburg, Germany, July 17-22, 2016*, Seiten 263–268. IEEE, 2016.

[PG04] PICARD, GAUTHIER und MARIE-PIERRE GLEIZES: *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Kapitel The ADELFE Methodology, Seiten 157–175. Springer US, Boston, MA, 2004.

[PH05] PARASHAR, MANISH und SALIM HARIRI: *Autonomic Computing: An Overview*. In: *Proceedings of the 2004 International Conference on Unconventional Programming Paradigms*, UPP'04, Seiten 257–269, Berlin, Heidelberg, 2005. Springer-Verlag.

[Pok07] POKAHR, ALEXANDER: *Entwicklung verteilter agentenorientierter Softwaresysteme: Programmiersprachen und Werkzeuge f ür intelligente Agenten*. Doktorarbeit, Fakultät für Mathematik, Informatik und Naturwissenschaften, Universität Hamburg, 2007.

[Pos81] POSTEL, JON: *Transmission control protocol*. RFC 793, RFC Editor, September 1981.

[PP04] POMBERGER, GUSTAV und WOLFGANG PREE: *Software Engineering Architektur-Design und Prozessorientierung*. Carl Hanser Verlag GmbH & Co. KG, 3. Auflage, 2004.

[PR12] PREISLER, THOMAS und WOLFGANG RENZ: *Scalability and Robustness Analysis of a Multi-Agent based Self-healing Resourceflow System*. In: GANZHA, MARIA, LESZEK A. MACIASZEK und MARCIN PAPRZYCKI (Herausgeber): *Federated Conference on Computer Science and Information Systems - FedCSIS 2012, Wroclaw, Poland, 9-12 September 2012, Proceedings*, Seiten 1261–1268, 2012.

[PR15] PREISLER, THOMAS und WOLFGANG RENZ: *Structural Adaptations for Self-Organizing Multi-Agent Systems*. In: *Seventh International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE), Nice, France*, 2015.

[PRD15] PREISLER, THOMAS, WOLFGANG RENZ und TIM DETHLEFS: *Structural Adaptation for Self-Organizing Multi-Agent Systems: Engineering and Evaluation*. International Journal on Advances in Intelligent Systems, 8(3&4):413–425, dec 2015.

[Pre13] PREISLER, THOMAS: *Policy-Based Approach for Non-Functional Requirements in Self-adaptive and Self-Organizing Systems*. In: KLUSCH, MATTHIAS, MATTHIAS THIMM und MARCIN PAPRZYCKI (Herausgeber): *Multiagent System Technologies - 11th German Conference, MATES 2013, Koblenz, Germany, September 16-20, 2013. Proceedings*, Band 8076 der Reihe *Lecture Notes in Computer Science*, Seiten 420–423. Springer, 2013.

[Pro13] PROKOPENKO, MIKHAIL: *Design Versus Self-Organization*. In: *Advances in Applied Self-Organizing Systems*. Springer London, 2013.

[PRV13] PREISLER, THOMAS, WOLFGANG RENZ und ANTE VILENICA: *Bike-sharing System Reliability Problems - A Data-based Analysis and Simulation Architecture*. In: WOLFGANG KERSTEN, THORSTEN BLECKER, CHRISTIAN M. RINGLE (Herausgeber): *Sustainability and Collaboration in Supply Chain Management: A Comprehensive Insight into Current Management Approaches*, Band 16 der Reihe *Supply Chain, Logistics and Operations Management*, Seiten 83–97. Josef Eul Verlag, 8 2013.

[PvdH07] PAPAZOGLOU, MIKE P. und WILLEM-JAN VAN DEN HEUVEL: *Service oriented architectures: approaches, technologies and research issues*. The VLDB Journal, 16(3):389–415, 2007.

[PVR13] PREISLER, THOMAS, ANTE VILENICA und WOLFGANG RENZ: *Decentralized Coordination in Self-Organizing Systems based on Peer-to-Peer Coordination Spaces*. Electronic Communications of the EASST, 56, 2013.

[RD01] ROWSTRON, ANTONY I. T. und PETER DRUSCHEL: *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*. In: *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, Seiten 329–350, London, UK, UK, 2001. Springer-Verlag.

[Res01] RESEARCH, IBM: *Autonomic Computing Manifesto: IBM's Perspective on the State of Information Technolo-*

*gy*. http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf, März 2001. Aufgerufen am: 26.02.2016.

[RG+95] RAO, ANAND S, MICHAEL P GEORGEFF und OTHERS: *BDI agents: From theory to practice.* In: *ICMAS*, Band 95, Seiten 312–319, 1995.

[RGS+13] RIVOLA, D., A. GIUSTI, M. SALANI, A. E. RIZZOLI, R. RUDEL und L. M. GAMBARDELLA: *A decentralized approach to demand side load management: The Swiss2Grid project*. In: *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, Seiten 4704–4709, Nov 2013.

[RJC12] RAMIREZ, ANDRES J., ADAM C. JENSEN und BETTY H. C. CHENG: *A Taxonomy of Uncertainty for Dynamically Adaptive Systems*. In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '12, Seiten 99–108, Piscataway, NJ, USA, 2012. IEEE Press.

[RKCD01] ROWSTRON, ANTONY, ANNE-MARIE KERMARREC, MIGUEL CASTRO und PETER DRUSCHEL: *Scribe: The Design of a Large-Scale Event Notification Infrastructure*, Seiten 30–43. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[RL05] ROBERTSON, PAUL und ROBERT LADDAGA: *Self-star Properties in Complex Information Systems: Conceptual and Practical Foundations*, Kapitel Model Based Diagnosis and Contexts in Self Adaptive Software, Seiten 112–127. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[RMB+06] RICHTER, URBAN, MOEZ MNIF, JÜRGEN BRANKE, CHRISTIAN MÜLLER-SCHLOER und HARTMUT SCHMECK: *Towards a generic observer/controller architecture for Organic Computing.* GI Jahrestagung (1), 93:112–119, 2006.

[RN04] RUSSEL, STUART und PETER NORVIG: *Künstliche Intelligenz - Ein moderner Ansatz*. Pearson Studium, 2. Auflage Auflage, 2004.

[Rot05] ROTH, JÖRG: *Mobile Computing - Grundlagen, Technik, Konzepte*. dpunkt.verlag, 2., aktualisierte Auflage Auflage, 2005.

[Roy87] ROYCE, W. W.: *Managing the Development of Large Software Systems: Concepts and Techniques*. In: *Proceedings of the 9th International Conference on Software Engineering*, ICSE '87, Seiten 328–338, Los Alamitos, CA, USA, 1987. IEEE Computer Society Press.

[RPS12] RENZ, WOLFGANG, THOMAS PREISLER und JAN SUDEIKAT: *Mesoscopic Stochastic Models for Validating Self-Organizing Multi-Agent Systems*. In: *Sixth IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, SASOW 2012, Lyon, France, September 10-14, 2012*, Seiten 119–126. IEEE Computer Society, 2012.

[RPV11] RICCI, ALESSANDRO, MICHELE PIUNTI und MIRKO VIROLI: *Environment programming in multi-agent systems: an artifact-based perspective*. Autonomous Agents and Multi-Agent Systems, 23(2):158–192, 2011.

[RR06] RESCORLA, ERIC und N RESONANCE: *Introduction to distributed hash tables*. IAB plenary, IETF, 65, 2006.

[RS09] RENZ, WOLFGANG und JAN SUDEIKAT: *Organized Adaption in Multi-Agent Systems: First International Workshop, OAMAS 2008, Estoril, Portugal, May 13, 2008. Revised and Invited Papers*, Kapitel Modeling Feedback within MAS: A Systemic Approach to Organizational Dynamics, Seiten 72–89. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[RV05] RICCI, ALESSANDRO und MIRKO VIROLI: *Coordination Artifacts: A Unifying Abstraction for Engineering Environment-Mediated Coordination in MAS*. Informatica (Slovenia), 29(4):433–444, 2005.

[RVO08] RICCI, ALESSANDRO, MIRKO VIROLI und ANDREA OMICINI: *Programming Multi-Agent Systems: 5th International Workshop, ProMAS 2007 Honolulu, HI, USA, May 15, 2007 Revised and Invited Papers*, Kapitel The A&A Programming Model and Technology for Developing Agent Environments in MAS, Seiten 89–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[RWM+14] ROHJANS, S., E. WIDL, W. MÜLLER, S. SCHÜTTE und S. LEHNHOFF: *Gekoppelte Simulation komplexer Energiesysteme mittels MOSAIK und FMI (Co-Simulation of Complex Energy Systems with MOSAIK and FMI)*. Automatisierungstechnik, 62(5), Mai 2014.

[RWvM10] REINECKE, PHILIPP, KATINKA WOLTER und AAD VAN MOORSEL: *Evaluating the Adaptivity of Computing Systems*. Perform. Eval., 67(8):676–693, August 2010.

[SBP+09] SUDEIKAT, JAN, LARS BRAUBACH, ALEXANDER POKAHR, WOLFGANG RENZ und WINFRIED LAMERSDORF: *Systematically Engineering Self-Organizing Systems : The SodekoVS Approach*. Electronic Communications of the EASST, 17:12, 2009.

[SGK06] SERUGENDO, GIOVANNA DI MARZO, MARIE-PIERRE GLEIZES und ANTHONY KARAGEORGOS: *Self-organisation and emergence in MAS: An overview*. Informatica, 30(1):45–54, 2006.

[SH04] SCHELFTHOUT, KURT und TOM HOLVOET: *An Environment for Coordination of Situated Multi-Agent Systems*. In: *1st Int. Workshop on Environments for Multi-Agent Systems*, 2004.

[Sha48] SHANNON, CLAUDE E.: *A mathematical theory of communication*. Bell System Tech. J., 27:623–656, 1948.

[Sha01] SHALIZI, C. R.: *Causal Architecture, Complexity, and Self-Organization in Time Series and Cellular Automata*. Doktorarbeit, Physics Department, University of Wisconsin-Madison, 2001.

[SHB97] SCHOONDERWOERD, RUUD, OWEN HOLLAND und JANET BRUTEN: *Ant-like Agents for Load Balancing in Telecommunications Networks*. In: *Proceedings of the First International Conference on Autonomous Agents*, AGENTS '97, Seiten 209–216, New York, NY, USA, 1997. ACM.

[SL01] SCHEUTZ, MATTHIAS und BRIAN LOGAN: *Affective vs. Deliberative Agent Control*. In: *Proceedings of the Artificial Intelligence and the Simulation of Behaviour Symposium on Emotion, Cognition and Affective Computing*, Seiten 1–10, 2001.

[Sos11] SOSINSKY, BARRIE: *Cloud Computing Bible*. Wiley, 2011.

[SPTU05] STERRITT, ROY, MANISH PARASHAR, HUAGLORY TIANFIELD und RAINER UNLAND: *A Concise Introduction to Autonomic Computing*. Adv. Eng. Inform., 19(3):181–187, Juli 2005.

[SR08] SUDEIKAT, JAN und WOLFGANG RENZ: *Applications of Complex Adaptive Systems*, Kapitel Building Complex Adaptive Systems: On Engineering Self-Organizing Multi-Agent Systems, Seite 28. Hershey: IGI Global, 2008.

[SR09a] SUDEIKAT, J. und W. RENZ: *DeCoMAS: An Architecture for Supplementing MAS with Systemic Models of Decentralized Agent Coordination*. In: *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT '09. IEEE/WIC/ACM International Joint Conferences on*, Band 2, Seiten 104–107, Sept 2009.

[SR09b] SUDEIKAT, JAN und WOLFGANG RENZ: *MASDynamics: Toward Systemic Modeling of Decentralized Agent Coordination*. In: DAVID, KLAUS und KURT GEIHS (Herausgeber): *Kommunikation in Verteilten Systemen (KiVS)*, Informatik aktuell, Seiten 79–90. Springer Berlin Heidelberg, 2009.

[SR10a] SUDEIKAT, JAN und WOLFGANG RENZ: *On the Modeling, Refinement and Integration of Decentralized Agent Coordination: A Case Study on Dissemination Processes in Networks*. In: *Proceedings of the First International Conference on Self-organizing Architectures*, SOAR'09, Seiten 251–274, Berlin, Heidelberg, 2010. Springer-Verlag.

[SR10b] SUDEIKAT, JAN und WOLFGANG RENZ: *Systemic Modeling of Agent Coaction: A Catalog of Decentralized Coordinating Processes*. Electronic Communications of the EASST, 27, 2010.

[SR11] SUDEIKAT, JAN und WOLFGANG RENZ: *Agent-Oriented Software Engineering X: 10th International Workshop, AOSE 2009, Budapest, Hungary, May 11-12, 2009, Revised Selected Papers*, Kapitel Qualitative Modeling of MAS Dynamics, Seiten 80–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[SS12] SCHERFKE, S. und S. SCHÜTTE: *mosaik - Architecture Whitepaper*. Technischer Bericht, OFFIS, 2012.

[SSS+10a] SUDEIKAT, JAN, JAN-PHILIPP STEGHÖFER, HELLA SEEBACH, WOLFGANG REIF, WOLFGANG RENZ, THOMAS PREISLER und PETER SALCHOW: *Design and Simulation of a Wave-like Self-Organization Strategy for Resource-Flow Systems*. In: BOISSIER, OLIVIER, AMAL EL FALLAH-SEGHROUCHNI, SALIMA HASSAS und NICOLAS MAUDET (Herausgeber): *Proceedings of The Multi-Agent Logics, Languages, and Organisations Federated Workshops (MALLOW 2010), Lyon, France, August 30 - September 2, 2010*, Band 627 der Reihe *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[SSS+10b] SUDEIKAT, JAN, JAN-PHILIPP STEGHÖFER, HELLA SEEBACH, WOLFGANG REIF, WOLFGANG RENZ, THOMAS PREISLER und PETER SALCHOW: *A Wave-like Decentralized Reconfiguration Strategy for Self-organizing Resource-Flow Systems (Short Paper)*. Electronic Proceedings in Theoretical Computer Science, 27:32–33, 2010. Proceedings of the Workshop.

[SSS+12] SUDEIKAT, J., J. P. STEGHÖFER, H. SEEBACH, W. REIF, W. RENZ, T. PREISLER und P. SALCHOW: *On the Combination of Top-down and Bottom-up Methodologies for the Design of Coordination Mechanisms in Self-organising Systems*. Information and Software Technology, 54(6):593–607, Juni 2012.

[ST09] SALEHIE, MAZEIAR und LADAN TAHVILDARI: *Self-adaptive Software: Landscape and Research Challenges*. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 4(2):14:1–14:42, Mai 2009.

[Ste00] STERMAN, JOHN D.: *Business Dynamics - Systems Thinking and Modeling for a Complex World*. McGraw-Hill, 2000.

[Sud10] SUDEIKAT, JAN: *Engineering Self-Organizing Dynamics in Distributed Systems: A Systemic Approach*. Doktorarbeit, Universität Hamburg, Fachbereich Informatik, Verteilte Systeme und Informationssysteme, 2010.

[SWA+13] STIFTER, M., E. WIDL, F. ANDREN, A. ELSHEIKH, T. STRASSER und P. PALENSKY: *Co-simulation of components, controls and power systems based on open source software*. In: *2013 IEEE Power Energy Society General Meeting*, Seiten 1–5, July 2013.

[Tab06] TABONY, J.: *Historical and conceptual background of self-organization by reactive processes*. Biol Cell., 98(10):589–602, 2006.

[TB90] TRIPP, STEVEN D. und BARBARA BICHELMEYER: *Rapid prototyping: An alternative instructional design strategy*. Educational Technology Research and Development, 38(1):31–44, 1990.

[TT09] TAYLOR, ANNETTE F. und MARK R. TINSLEY: *Chemical self-organization: A path to patterns*. Nature Chemistry, 1:340–341, 2009.

[TvS07] TANENBAUM, ANDREW und MAARTEN VAN STEEN: *Verteilte Systeme - Prinzipien und Paradigmen*. Pearson Studium, 2. Auflage Auflage, 2007.

[TW04] TISUE, SETH und URI WILENSKY: *Netlogo: A simple environment for modeling complexity*. In: *In International Conference on Complex Systems*, Seiten 16–21, 2004.

[UW09] UHRMACHER, ADELINDE M. und DANNY WEYNS: *Multi-Agent Systems: Simulation and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st Auflage, 2009.

[VCO09] VIROLI, MIRKO, MATTEO CASADEI und ANDREA OMICINI: *A Framework for Modelling and Implementing Self-organising Coordination*. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, Seiten 1353–1360, New York, NY, USA, 2009. ACM.

[VDE03] VDE/ITG/GI: *Organic Computing: Computer- und Systemarchitektur im Jahr 2010 - Positionspapier*. Positionspapier, Gesellschaft für Informatik (GI) und Informationstechnische Gesellschaft im VDE (ITG), 2003.

[VDPB04] VAN DYKE PARUNAK, H. und SVEN A. BRUECKNER: *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, Kapitel Engineering

Swarming Systems, Seiten 341–376. Springer US, Boston, MA, 2004.

[VDPBS02] VAN DYKE PARUNAK, H., S. BRUECKNER und J. SAUTER: *Digital pheromone mechanisms for coordination of unmanned vehicles*. In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, Seiten 449–450. ACM, 2002.

[vHadA02] HEE, KEES VAN und WIL M. P. AN DER AALST: *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2002.

[VHL+11] VILENICA, ANTE, KRISTOF HAMANN, WINFRIED LAMERSDORF, JAN SUDEIKAT und WOLFGANG RENZ: *Distributed Applications and Interoperable Systems: 11th IFIP WG 6.1 International Conference, DAIS 2011, Reykjavik, Iceland, June 6-9, 2011. Proceedings*, Kapitel An Extensible Framework for Dynamic Market-Based Service Selection and Business Process Execution, Seiten 150–164. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[Vil13] VILENICA, ANTE: *Anwendungsentwicklung selbstorganisierender Systeme: Systematische Konstruktion und Evaluierung*. Doktorarbeit, Fakultät für Mathematik, Informatik und Naturwissenschaften, Universität Hamburg, 2013.

[VL11] VILENICA, ANTE und WINFRIED LAMERSDORF: *Simulation Management for Agent-Based Distributed Systems*, Seiten 477–492. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[VL12] VILENICA, ANTE und WINFRIED LAMERSDORF: *Benchmarking and Evaluation Support for Self-Adaptive Distributed Systems*. In: *Proceedings of the 2012 Sixth International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, CISIS '12, Seiten 20–27, Washington, DC, USA, 2012. IEEE Computer Society.

[vLH12] LON, R.R.S. VAN und T. HOLVOET: *RinSim: A Simulator for Collective Adaptive Systems in Transportation and Logistics*. In: *Self-Adaptive and Self-Organizing Systems (SASO), 2012 IEEE Sixth International Conference on*, Seiten 231–232, Sept 2012.

[VM10] VOGEL, P. und D.C. MATTFELD: *Modeling of Repositioning Activities in Bike-sharing Systems*. In: *World Conference on Transport Research (WCTR)*, Lissabon, 2010.

[VRSL09] VILENICA, A., W. RENZ, J. SUDEIKAT und W. LAMERSDORF: *Multi-Agent-Architecture for simulating traffic management - A*

*case study on highway networks*. In: *2nd International Workshop on Nonlinear Dynamics and Synchronization*, Seiten 121–127, July 2009.

[VS09] VILENICA, ANTE und JAN SUDEIKAT: *Methods and Tools for Engineering Self-Organizing Software Systems*. Electronic Communications of the EASST, 37, 2009.

[VSL+10] VILENICA, ANTE, JAN SUDEIKAT, WINFRIED LAMERSDORF, WOLFGANG RENZ, LARS BRAUBACH und ALEXANDER POKAHR: *Coordination in Multi-Agent Systems: A Declarative Approach using Coordination Spaces*. In: TRAPPL, R. (Herausgeber): *Proc. of the 20th European Meeting on Cybernetics and Systems Research (EMCSR 2010) - Int. Workshop From Agent Theory to Agent Implementation (AT2AI-7)*, Seiten 441–446. Austrian Society for Cybernetic Studies, 4 2010.

[WH04] WOLF, TOM DE und TOM HOLVOET: *Emergence and Self-Organisation: a statement of similarities and differences*. In: *Proceedings of the International Workshop on Engineering Self-Organising Applications*, Seiten 96–110. Springer, 2004.

[WH06] WOLF, TOM DE und TOM HOLVOET: *A Catalogue of Decentralised Coordination Mechanisms for Designing Self-Organising Emergent Applications*. Technischer Bericht CW 458, Deptartment of Computer Science, K.U. Leuven, 2006.

[Whi97] WHITE, T.: *Swarm intelligence and problem solving in telecommunications*. Canadian Artificial Intelligence Magazin, 41:14–16, 1997.

[WJ95] WOOLDRIDGE, MICHAEL und NICHOLAS R. JENNINGS: *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review, 10:115–152, 1995.

[WMA10] WEYNS, DANNY, SAM MALEK und JESPER ANDERSSON: *On Decentralized Self-adaptation: Lessons from the Trenches and Challenges for the Future*. In: *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '10, Seiten 84–93, New York, NY, USA, 2010. ACM.

[Wol07] WOLF, TOM DE: *Analysing and engineering self-organising emergent applications*. Doktorarbeit, Informatics Section, Department of Computer Science, Faculty of Engineering Science, Catholic University Leuven, 2007.

[Woo09] WOOLDRIDGE, MICHAEL: *An Introduction to Multi Agent Systems*. Wiley, 2. Auflage Auflage, 2009.

[WSR+05] WEYNS, DANNY, MICHAEL SCHUMACHER, ALESSANDRO RICCI, MIRKO VIROLI und TOM HOLVOET: *Environments in multi-agent systems*. The Knowledge Engineering Review, 20(2):127–141, 2005.

[WvdPdM03] WEIGAND, H., F. VAN DER POLL und A. DE MOOR: *Coordination through Communication*. In: H.WEIGAND, G. GOLDKUHL und A. DE MOOR (Herausgeber): *Proceedings of the 8th International Working Conference on the Language-Action Perspective on Communication Modelling*, 2003.

[XSY+05] XU, YANG, PAUL SCERRI, BIN YU, STEVEN OKAMOTO, MICHAEL LEWIS und KATIA SYCARA: *An Integrated Token-based Algorithm for Scalable Coordination*. In: *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, Seiten 407–414, New York, NY, USA, 2005. ACM.

[YÖ09] YILMAZ, LEVENT und TUNCER ÖREN (Herausgeber): *Agent-Directed Simulation and Systems Engineering*. Wiley, September 2009.

[Zad63] ZADEH, LOTFI A.: *On the definition of adaptivity*. Proceedings of the IEEE, 51(3):469–470, 1963.

[ZJW03] ZAMBONELLI, FRANCO, NICHOLAS R. JENNINGS und MICHAEL WOOLDRIDGE: *Developing Multiagent Systems: The Gaia Methodology*. ACM Trans. Softw. Eng. Methodol., 12(3):317–370, Juli 2003.

[Zno06] ZNOTKA, JÜRGEN: *Autonome Agenten: Einsatz bei der Erstellung eines Stundenplans*. In: *11. Norddeutsches Kolloquium für Informatik an Fachhochschulen*, 2006.

[ZR14] ZACHER, SERGE und MANFRED REUTER: *Regelungstechnik für Ingenieure: Analyse, Simulation und Entwurf von Regelkreisen*. Springer Vieweg, 14 Auflage, 2014.

[ZVDP03] ZAMBONELLI, FRANCO und H. VAN DYKE PARUNAK: *Towards a Paradigm Change in Computer Science and Software Engineering: A Synthesis*. Knowl. Eng. Rev., 18(4):329–342, Dezember 2003.

[ZZyDB15] ZHANG, LIHONG, JUN ZHANG, ZHENG YU DUAN und DAVID BRYDE: *Sustainable bike-sharing systems: characteristics and commonalities across cases in urban China*. Journal of Cleaner Production, 97:124 – 133, 2015. Special Volume: Why have Sustainable Product-Service Systems not been widely implemented?

## Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Hamburg, den 1. Februar 2017

Thomas Preisler