UNIVERSITÄT ZU LÜBECK
INSTITUTE FOR SOFTWARE ENGINEERING
AND PROGRAMMING LANGUAGES

Ein Szenarien-Simulator für Selbst-Optimierende Emergente Systeme

A Simulator for Scenarios of Self-Optimizing Emergent Systems

Masterarbeit

im Rahmen des Studiengangs
Entrepreneurship in digitalen Technologien
der Universität zu Lübeck

vorgelegt von
Simon Paasche

ausgegeben und betreut von
Prof. Dr. Martin Leucker

mit Unterstützung von
Prof. Dr. Klaus Schmid
Christian Kröher

Die Masterarbeit ist im Rahmen einer Tätigkeit in der Arbeitsgruppe Software Systems Engineering der Stiftung Universität Hildesheim entstanden.

Lübeck, den 25. November 2020

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne die Benutzung anderer als der angegebenen Hilfsmittel selbständig verfasst habe; die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe des Literaturzitats gekennzeichnet.

(Simon Paasche)
Lübeck, den 25. November 2020

Kurzfassung Das Internet-of-Things vernetzt zunehmend die Gesellschaft und Wirtschaft. Viele smarte Komponenten interagieren in dieser Umwelt miteinander und tauschen Informationen (z.B Sensordaten) aus, um eine gemeinsame, höhere Aufgabe zu erledigen. Smarte Komponenten handeln dabei unabhängig voneinander und bilden so eigenständige Teilsysteme. Durch das Zusammenspiel dieser unabhängigen Teilsysteme ensteht ein komplexes Gesamtsystem. Ein Beispiel für ein solches komplexes Gesamtsystem ist ein smartes Electric-Grid, welches unter anderem aus vielen kleinen Verbrauchern und Erzeugern besteht. Treten in einem Electric-Grid Veränderungen auf, erhöht sich beispielsweise der Verbrauch einer Komponente, reagiert das System auf diese Änderungen und ist in der Lage, eigenständig Prozesse anzustoßen, damit das Gesamtsystem stabil gehalten wird. In dem Fall eines Electric-Grids könnten die Prozesse so aussehen, dass einer der Erzeuger seine Produktion vorübergehend erhöht. Obwohl Interaktionen wie diese spezifizierten Regeln folgen, kann während der Laufzeit eines komplexen Systems neues, sogenanntes emergentes Verhalten auftreten. Die genauen Ausprägungen dieses Verhaltens sind vorab nur schwer bestimmbar. Einen Ansatz für die Abschätzung möglicher Emergenzen bieten Simulationsverfahren. Aus diesem Grund thematisiert die Masterarbeit die Entwicklung eines Konzepts für einen softwarebasierten Simulator für IoT-Systeme. Das Ziel des softwarebasierten Ansatzes ist ein kurzes Feedbacksystem zur Validierung der spezifizierten Regeln und Verhaltensweisen.

Abstract The Internet of Things is increasingly networking society and business. Many smart components interact with each other in this environment and exchange information (e.g. sensor data) to perform a common, higher task. Smart components act independently of each other and thus form independent subsystems. The interaction of these independent subsystems results in a complex overall system. One example of such a complex overall system is a smart electric grid, which consists of many small consumers and generators. If changes occur in an electric grid, for example, if the consumption of a component increases, the system reacts to these changes and is able to initiate processes independently to keep the overall system stable. In the case of an Electric Grid, the processes could be such that one of the producers temporarily increases its production. Although interactions like these follow specified rules, new, so-called emergent behavior can occur during the runtime of a complex system. The exact characteristics of this behavior are difficult to determine in advance. Simulation methods offer an approach for the estimation of possible emergences. For this reason, this master thesis deals with the development of a concept for a software-based simulator for IoT systems. The goal of the software-based approach is a short feedback system to validate the specified rules and behavior.

Inhaltsverzeichnis

1. Einleitung

Der vielfache Einsatz und die zunehmende Vernetzung smarter Geräte gestalten die Welt stetig komplexer und haben längst Einzug in viele Bereiche der Wirtschaft, Industrie und Gesellschaft gefunden. Durch die Integration dieser smarten Geräte können Produktionsabläufe und Prozesse in Unternehmen automatisiert und optimiert oder Menschen in ihrem Alltag unterstützt werden. Dadurch werden Unternehmen wettbewerbsfähiger und befähigt, wichtige Ressourcen optimal zu nutzen. Diese gravierenden Veränderungen der Umwelt betreffen auch die Versorgungsbetriebe, beispielsweise Stromlieferanten oder Wasserwerke. Die Vision eines smarten Electric-Grids im Fall der Stromversorgung ist, dass Geräte wie Verbraucher und Erzeuger autonom die Stromflüsse verhandeln und flexibel auf Änderungen reagieren können. Nach Paya und Marinescu [PM14] sprechen unter anderem eine höhere Zuverlässigkeit des Stromnetzes aufgrund automatischer Fehlererkennung und -behebung sowie eine effizientere Energieerzeugung und damit geringere Emissionen für den Einsatz smarter Geräte in einem Electric-Grid.

Aktuelle Electric-Grids, die auch in dieser Masterarbeit betrachtet werden, enthalten bereits heute viele smarte Komponenten, z.B. Sensoren wie intelligente Stromzähler. Durch diese smarten Komponenten können sicherheits- und zeitrelevante Daten schnell erfasst und verarbeitet werden, damit ein Gleichgewicht zwischen Stromerzeugung und -verbrauch hergestellt werden kann. Smarten Komponenten und somit auch Electric-Grids können dem Internet of Things (IoT) zugeordnet werden. Dies meint, dass die Komponenten über das Internet miteinander vernetzt sind und Daten austauschen können. Damit finden Interaktionen zwischen den sogenannten IoT-Devices statt. Durch das stetige Wachstum dieser zunehmend intelligenten und adaptiven Komponenten entwickeln sich Electric-Grids sowie andere IoT-Landschaften zu komplexen Ökosystemen, in denen emergentes Verhalten auftreten kann. Um den Einfluss von beispielsweise neuen Updates oder veränderten Arbeitskonfigurationen, die in das laufende System eingespielt werden sollen, besser abschätzen zu können, sollten die Systemänderungen vorab getestet werden. Die vorliegende Masterarbeit setzt dazu auf der IoT-Schicht an.

Durch eine gründliche Analyse wird versucht, die IoT-Umwelt mit ihren Komponenten und deren Verhaltensweisen zu erfassen und in einem Konzept abzubilden, mit welchem mögliche Szenarien eines Electric-Grids sowie verwandter Domänen modellierbar und simulierbar sind. Die Masterarbeit trägt daher den Titel *Ein Szenarien-*

Simulator für Selbstoptimierende Emergente Systeme und beschreibt die Erstellung eines Konzepts für die Realisierung eines softwarebasierten Simulators. Die Arbeit ist dabei thematisch dem Projekt *DevOpt* zugeordnet. Der Projektsteckbrief ist im Anhang unter Abschnitt A.1 auf Seite 81 zu finden. Das Projekt beschäftigt sich mit der Entwicklung und Erprobung kontrollierter, selbstorganisierender, emergenter Systeme. Das bedeutet, dass die Systeme unter der Kontrolle einer oder mehrerer übergeordneter Instanzen neues Verhalten herausbilden. Emergentes Verhalten ist dabei Verhalten, welches nicht explizit in dem Programmcode abgebildet ist, sondern durch das Zusammenspiel der (Teil-)Systeme und die Interaktionen der einzelnen smarten Komponenten entsteht und somit erst zur Laufzeit des Gesamtsystems sichtbar wird. Von Selbstorganisation bzw. Selbstoptimierung spricht man, wenn (Teil-)Systeme ohne den Einfluss einer zentralen Instanz flexibel auf veränderte Bedingungen reagieren können.

Bei der Modellierung eines komplexen IoT-Systems müssen während des Entwicklungsprozess einige Abstraktionen erfolgen. Daher soll in dieser Arbeit die wissenschaftliche Frage, inwiefern ein IoT-System modelliert und in einem softwarebasierten Simulator abgebildet werden kann, sodass potenzielles emergentes Verhalten während der Simulation zum Vorschein kommt, analysiert werden. Zudem wird der Frage nachgegangen, welche Einschränkungen man eingeht, wenn die veränderten Arbeitskonfigurationen und Updates softwarebasiert und somit nicht in dem realen System getestet werden.

1.1. Verwandte Arbeiten

In der Literatur finden sich viele Ansätze zur Realisierung eines softwarebasierten Simulators für IoT-Systeme. Die vorhandenen Simulatoren und Plattformen haben größtenteils gemeinsam, dass der Architektur eine Cloud zugrunde liegt, in der relevante Applikationen ablaufen und Verbindungen von der Cloud zu den IoT-Devices bestehen. Weiterhin eint die Ansätze, dass sie entwickelt wurden, um Szenarien des IoT zu testen. Aufgrund der Vielzahl an existierenden Ansätzen wird an dieser Stelle nur eine enge Auswahl erwähnt.

Dias et al. geben in ihrer Arbeit [DCPF18] einen guten Überblick über verschiedene Testansätze, beispielsweise Unit- und System-Tests, sowie aktuelle Plattformen und Simulatoren für IoT-Systeme. Eine Besonderheit bei den in [DCPF18] vorgestellten Simulatoren stellt MobIoTSim [PKSL16] dar. Der Simulator unterscheidet sich von anderen dadurch, dass es ein auf Android basierter, mobiler Simulator ist. Der Simulator ist in der Lage, mehrere IoT-Devices zu simulieren und die anfallenden Daten an eine Cloud zu senden. Im Vordergrund bei der Entwicklung von MobIoTSim stand nicht die Evaluation der gesamten IoT-Landschaft, sondern das

Testen von Cloud-Anwendungen mit einem mobilen Device, wie etwa einem Smartphone oder Tablet. Der Ansatz liefert einige gute Erkenntnisse in der Entwicklung der IoT-Devices, unterscheidet sich jedoch grundlegend von dieser Masterarbeit. In dieser Masterarbeit soll vordergründig die gesamte IoT-Ebene erfasst und simuliert werden. Die Kommunikation mit einer Cloud ist nicht von Interesse.

Lai et al. [LBBC19] präsentieren eine Architektur basierend auf Microservices. Die Microservices bieten eine sehr gute Skalier- und Erweiterbarkeit. Der Fokus der Arbeit liegt auf Agentenbewegungen innerhalb der IoT-Umwelt (z.B durch Elektrofahrzeuge). Dabei wird das Auftreten emergenten Verhaltens nicht näher betrachtet, welches jedoch in dieser Masterarbeit verstärkt in den Blick genommen wird.

YAFS [LGJ19] ist ein weiterer erst kürzlich entwickelter Ansatz aus dem Jahr 2019. Der Simulator ist dabei mittels Fog-Computing realisiert. In der Architektur sind auf der IoT-Schicht keine Hierarchien vorhanden. Die einzelnen IoT-Devices sind über ein Netzwerk aus Fog-Knoten mit der Cloud verbunden. Die Arbeit fokussiert unter anderem die Topologie des Netzwerks sowie Routing. Diese Punkte unteschеiden sich zu dem Simulator-Konzept dieser Masterarbeit. Durch die Gruppierung in Häuser und Stadtteile sowie die Einbindung einer Kontroll-Schicht sind Hierarchien auf der IoT-Schicht vorhanden. Da alle Berechnungen und Anwendungen lokal laufen, spielen die Netzwerktopologie und Routing keine Rolle.

Zudem erwähnenswert ist der Ansatz von Karnouskos und Holland [KdH09]. In ihrer Arbeit beschreiben Karnouskos und Holland eine dreistufige Architektur bestehend aus einer Anwendungs-, Logik- und Simulator-Schicht. Die unterste Schicht, die Simulator-Schicht, enthält die mittels des Java Agent Development (JADE) Framework umgesetzten Agenten. In dem beschrieben Ansatz werden Haushalte, Geräte, Fahrzeuge, Städte und Kraftwerk als Entitäten definiert, die während des aktiven Zustands Aktionen ausführen. Die Aktionen erfolgen anhand von vorgegebenen Energieprofilen und Regeln, die in der Logik-Schicht hinterlegt sind. Über die Anwendungs-Schicht können Änderungen in den laufenden Simulator eingebracht werden. Diese Masterarbeit unterscheidet sich von [KdH09], indem von spezifischen, vorgegebenen Energieprofilen abstrahiert wird, um das Agentverhalten möglichst frei zu halten. Weiterhin ist die Logik direkt in den Agenten definiert, um Autonomie und Unabhängigkeit zu erreichen.

Die Erweiterung einer vorhandenen Plattform bzw. eines existierenden Simulator-Konzepts hat sich als schwierig herausgestellt. Die Schwierigkeiten lagen zum einen in der Vielzahl der vorhandenen Ansätze, bei denen meist Implementierungen fehlten. Zum anderen lagen die Schwierigkeiten in den abweichenden Schwerpunkten der existierenden Ansätze. Die Ansätze fokussieren auf die prototypische Umsetzung und das Testen einer Smarten Stadt bzw. allgemein einer IoT-Landschaft, jedoch nicht auf der Abschätzung emergenten Verhaltens. Damit bot auch keiner der gefundenen Ansätze die Möglichkeit zur Integration einer mehrstufigen Kontroll-Schicht.

1.2. Aufbau der Arbeit

Neben dieser Einleitung und einer Zusammenfassung am Ende gliedert sich diese Arbeit in fünf Kapitel, die im Folgenden kurz skizziert werden.

Kapitel 2 beschreibt die für diese Arbeit benötigten Grundlagen. In diesem Kapitel werden exemplarisch drei mögliche Szenarien eines Electric-Grids vorgestellt und der jeweilige Ablauf ausführlich beschrieben. Im weiteren Verlauf klärt das Kapitel die Begriffe *Selbstorganisation* und *Emergenz* und beschreibt die zugrunde liegende Theorie. Zudem stellt das Kapitel mit der *Agentenbasierten Modellierung und Simulation* eine geeignete Herangehensweise für die Modellierung selbstorganisierender, emergenter Systeme vor.

Kapitel 3 bildet den Kern der Arbeit und stellt den Entwicklungsprozess des Simulator-Konzepts vor. Das Kapitel gliedert sich dazu in fünf Abschnitte. Diese beschreiben (1) die Analyse der Szenarien eines Electric-Grids, (2) die Definition von Anforderungen an das Konzept, (3) die agentenbasierte Modellierung der smarten Komponenten und ihrer Umwelt, (4) das für den Simulationsablauf benötigte Kommunikationsprotokoll und (5) die Gesamtarchitektur des Simulator-Konzepts. Das fertige Konzept wird später für die Realisierung eines prototypischen, softwarebasierten Simulators genutzt.

Kapitel 4 präsentiert die prototypische Implementierung des vorgestellten Konzepts. Dabei wird zu Beginn die Wahl der Programmiersprache und der Vorgehensweise begründet dargelegt. Im weiteren Verlauf werden relevante technische Umsetzungen, etwa die Abbildung des Agentenverhaltens und die Einbindung einer Kontroll-Schicht, beschrieben.

Kapitel 5 beinhaltet eine Evaluation des Simulator-Konzepts aus dem dritten Kapitel. In diesem Kapitel wird analysiert, welche Anforderungen das theoretische Konzept erfüllt. Für die Evaluation werden die zwei realisierten Prototypen zur Hilfe genommen. Dabei soll geklärt werden, welchen Funktionsumfang die Prototypen mitbringen und wie dieser die Anforderungen stützt oder eventuell verletzt. Am Ende des Kapitels wird zudem kurz auf die Entwicklung automatisierter Tests für die Prototypen eingegangen.

Kapitel 6 diskutiert die wichtigsten Kritikpunkte des vorgestellten Konzepts. Dazu werden zu Beginn des Kapitels die Einschränkungen des Simulator-Konzepts dargelegt und ein weiteres Feedbacksystem als Alternative zu der softwarebasierten Simulation beschrieben. Weiterhin thematisiert das Kapitel detailliert die Erweiterbarkeit des entwickelten Konzepts und stellt die Domänenunabhängigkeit des gewählten Ansatzes heraus.

2. Grundlagen und Szenarien

In diesem Kapitel werden die Grundlagen, die für das weitere Verständnis dieser Masterarbeit nötig sind, vorgestellt. Der erste Abschnitt des Kapitels befasst sich daher mit prototypischen Szenarien der smarten Energieversorgung. Die Beschreibung der Szenarien umfasst die verschiedenen Abstraktionsebenen eines Electric-Grids und erläutert die Abläufe innerhalb einer Ebene. Dafür wird jeweils ein für diese Ebene charakteristisches Beispiel herangezogen und anhand einer schematischen Abbildung vorgestellt. Das zu entwickelnde Simulator-Konzept soll später in der Lage sein, die Abstraktionsebenen zu modellieren und zu simulieren. Somit muss in den Szenarien herausgestellt werden, welche Grundlagen außerdem benötigt werden. Zu diesen Grundlagen gehören zum einen Wissen über Selbstorganisation und Emergenz in Systemen. Hierzu rekapituliert das Kapitel in dem zweiten Abschnitt die zugehörigen Definitionen, stellt die relevanten Konzepte vor und diskutiert die Verwendung dieser Ansätze. Zum anderen gehören zu den Grundlagen methodische Vorgehensweisen. Der dritte Abschnitt stellt eine geeignete Methode zur Modellierung eines selbstorganisierenden, emergenten Systems vor und begründet die Auswahl dieser Methode gegenüber Alternativen. Mittels der vorgestellten Methode kann ein agentenbasiertes Modell erstellt werden.

2.1. Projekt-Szenarien

In diesem Abschnitt werden drei mögliche Szenarien eines Electric-Grids jeweils anhand einer schematischen Darstellung beschrieben und exemplarisch durchlaufen. Die Abbildungen stellen dabei eine mögliche Ausprägung des jeweiligen Grids dar. Die Szenarien sind an die Prototypenszenarien des Projekts *DevOpt* angelehnt.

Wie in der Einleitung bereits erwähnt, ist die vorliegende Masterarbeit thematisch dem Projekt *DevOpt* zugeordnet. Ziel des Projekts ist die Entwicklung kontrollierter, emergenter Systeme. Ein emergentes System wird dabei, wie in Abbildung 2.1 auf der nächsten Seite abgebildet, als ein dreistufiges Schichtenmodell aufgefasst. Die drei Schichten des Modells sind die IoT-, die Kontroll- und die DevOps-Schicht. Als möglicher Anwendungskontext wurde ein Electric-Grid-Szenario gewählt. In einem solchen Szenario finden sich auf der IoT-Ebene beispielsweise smarte Häuser und

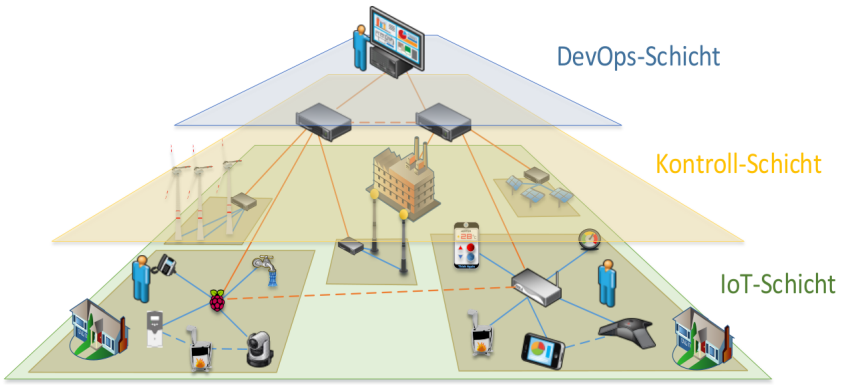


Abbildung 2.1.: Schematische Darstellung eines emergenten Systems. Ein emergentes System wird in dem Projekt *DevOpt* als dreistufiges, hierarchisches Schichtenmodell aufgefasst. Die drei Schichten sind die IoT-, die Kontroll- und die DevOps-Schicht. Auf der untersten Ebene interagieren die IoT-Devices einer Gruppe miteinander, um das individuelle Ziel (hier: die Verhandlung von lokalen Stromflüssen) zu erreichen. Die übergeordnete Kontroll-Schicht stellt die Verbindung für gruppenübergreifende Interaktionen her und greift gegebenenfalls zwecks Optimierung ein. Von der DevOps-Schicht aus kann der Anwender in das laufende System eingreifen und so beispielsweise neue Verbrauchsmuster oder Software-Updates einspielen. Die Abbildung ist der Projektbeschreibung aus Abschnitt A.1 auf Seite 81 entnommen.

Straßen. Diese mit intelligenter Technik ausgestatteten Instanzen verhandeln untereinander ihre Ressourcennutzung und Arbeitskonfigurationen. Die Kontroll-Schicht ist mehreren lokalen IoT-Systemen übergeordnet und verwaltet übergreifende Entscheidungen und Optimierungen. Auf der obersten Schicht, der DevOps-Schicht, wird das Gesamtsystem überwacht und es können von hier aus neue Programme und insbesondere Updates eingespielt oder Geräte aus der Ferne analysiert werden.

Die nachfolgenden Szenarien schildern mögliche Situationen, welche während des Betriebs eines smarten Electric-Grids auftreten können. In den drei Szenarien werden verschiedene Abstraktionslevel eingenommen. Diese sind: (1) ein Nano-Grid, (2) ein Micro-Grid und (3) ein Urban-Grid. Ein Nano-Grid stellt die feinste Ebene dar und fokussiert die Kommunikation und Abläufe der einzelnen IoT-Devices innerhalb eines smarten Hauses. Ein Micro-Grid abstrahiert von individuellen IoT-Devices und sieht das smarte Haus als kleinste Einheit an. Auf dieser Abstraktionsebene wird die Kommunikation innerhalb eines Stadtteils betrachtet. Ein Urban-Grid fügt mehrere Stadtteile zu einer smarten Stadt zusammen. Im Folgenden werden die einzelnen Grids fokussiert und nacheinander vorgestellt.

2.1.1. Nano-Grid

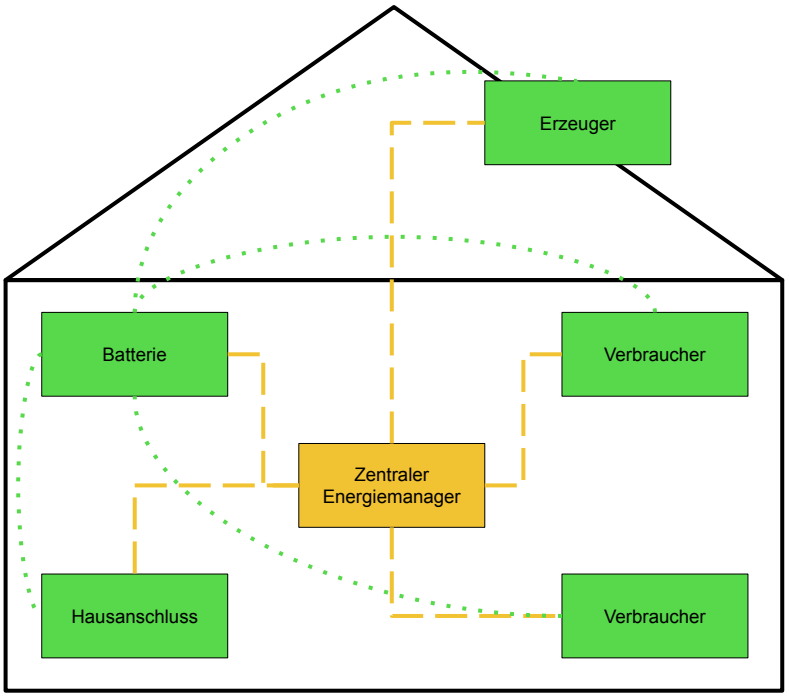


Abbildung 2.2.: Nano-Grid: Stromversorgung eines Einfamilienhauses. Das smarte Haus in der Abbildung besteht aus fünf Elementen der IoT-Schicht (grün) und einem Element der Kontroll-Schicht (orange). Die fünf IoT-Devices verhandeln autonom interne Stromflüsse und kommunizieren dabei direkt miteinander. Diese direkte Kommunikation ist beispielhaft für die Batterie eingezeichnet (grün gepunktet). Bei Unstimmigkeiten kann der Zentrale Energiemanager in die Verhandlungen eingreifen und die Stromflüsse selbstständig organisieren. Die Kontrolle ist über die gelb gestrichelte Linie visualisiert.

Ein Nano-Grid umfasst die kleinste Gruppierung von IoT-Devices. In einem Electric-Grid finden sich auf dieser Ebene die einzelnen IoT-Devices eines smarten Gebäudes. Hierzu zählen je nach Aufbau unter anderem Wohnhäuser, Fabriken und Krankenhäuser. Die Aufgaben, die die IoT-Devices auf dieser Ebene auszuführen haben, sind die lokale Sicherstellung der Stromversorgung und die Verwendung des eigens produzierten Stroms. Das bedeutet, es müssen einerseits die Verbraucher des Gebäudes mit Strom versorgt werden. Der Strom kann dabei aus internen Quellen oder dem öffentlichen Stromnetz stammen. Andererseits muss der eigens erzeugte Strom intern verbraucht oder in das öffentliche Stromnetz eingespeist werden. Die IoT-Devices verhandeln die Stromflüsse ohne Eingriff von außen und kommunizieren dabei direkt miteinander. Die lokale Kontroll-Schicht, genannt Zentraler Energiemanager, überwacht die Verhandlungen und justiert gegebenenfalls nach.

Abbildung 2.2 auf der vorherigen Seite zeigt, beispielhaft für ein Nano-Grid, ein smartes Wohnhaus. Das Haus besitzt einen Erzeuger, eine Batterie, zwei Verbraucher, einen Hausanschluss (Steckdose) und einen Energiemanager. Typische Verbraucher eines Nano-Grid sind Haushaltsgeräte, Fernseher, Beleuchtung und andere in Gebäuden vorkommende Elektrogeräte. In den Prototypenszenarien werden die Verbraucher als naiv dargestellt und haben jeweils einen Smart Plug vorgeschaltet, welcher für jegliche Interaktionen zuständig ist. Dies folgt der Annahme, dass Vebraucher nicht zwingend ab Werk smart sein müssen. Ohne Beschränkung der Allgemeinheit wird auf diese Annahme verzichtet und ein smarter Verbraucher als eine Instanz modelliert. Die grün hinterlegten Elemente stehen für Instanzen der lokalen IoT-Schicht. Orange hinterlegt sind die Elemente der lokalen Kontroll-Schicht. Ein mögliches Szenario in einem Nano-Grid sieht wie folgt aus: Der Erzeuger, z.B. eine Photovoltaikanlage (PVA), versorgt das Haus mit Strom. Ein Verbraucher ist aktiv und die produzierte Strommenge deckt den Bedarf optimal. Ein Bewohner schaltet einen zweiten Verbraucher ein. Der smarte Verbraucher fragt innerhalb der umliegenden IoT-Devices nach Strom. Da die PVA keine überschüssigen Ressourcen zur Verfügung hat, lehnt sie die Anfrage ab. Die Batterie übernimmt die Versorgung. Aufgrund zunehmender Sonneneinstrahlung nimmt die Stromproduktion der PVA zu. Der Energiemanager greift zwecks Optimierung in das Geschehen ein und veranlasst, dass nun auch der zweite Verbraucher von der PVA versorgt wird und die Batterien geladen werden. Nachdem die Batterien aufgeladen sind, lösen sie die Verbindung zur PVA. Die PVA fragt innerhalb der umliegenden IoT-Devices nach Abnehmern für den überschüssigen Strom. Mittels des Hausanschlusses wird dann der Strom in das öffentliche Netz eingespeist.

2.1.2. Micro-Grid

Als Micro-Grid bezeichnet man die nächst gröbere Betrachtungsebene. In einem Micro-Grid werden mehrere Nano-Grids zu einem smarten Straßenzug bzw. Stadtteil zusammengefasst. Das resultierende Micro-Grid dient dabei der gruppenübergreifenden Kommunikation zwischen den einzelnen smarten Gebäuden. Die Kommunikation zwischen den Nano-Grids erfolgt jeweils über die Zentralen Energiemanager. Die Energiemanager können direkt miteinander interagieren. So kann im Fall einer Stromüberproduktion ein benachbartes Haus versorgt werden. Die Interaktionen in einem Micro-Grid werden, analog zu den IoT-Devices eines Nano-Grids, von einer höheren Instanz überwacht und es kann bei Bedarf eingegriffen werden. Die zuständige Kontroll-Schicht heißt Sekundäre Schaltanlage (*Secondary Substation Node*, SNN). Dem SNN ist ein Management-System übergeordnet. Über dieses System können den Stadtteil betreffende Updates oder geänderte Verbrauchsmuster in das System eingespielt werden.

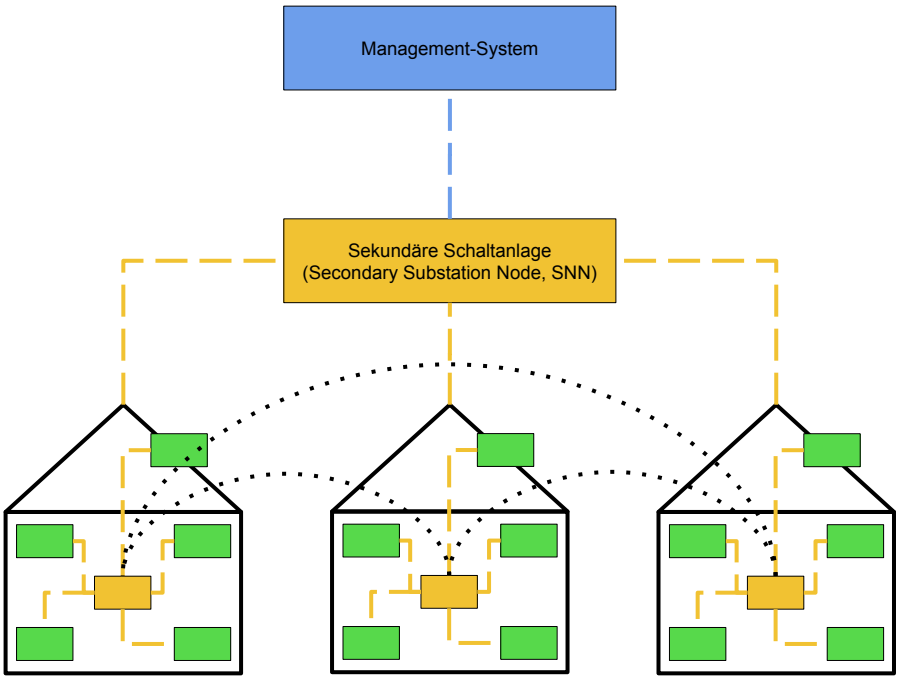


Abbildung 2.3.: Micro-Grid: Stromversorgung eines Stadtteils. Der Stadtteil besteht aus drei smarten Häusern. Analog zu den Interaktionen in einem Nano-Grid können die drei Häuser direkt miteinander kommunizieren. Die Kommunikation findet über die jeweiligen Energiemanager statt und ist über die gepunkteten Linien dargestellt. Die übergeordnete Kontroll-Schicht (orange), in Form des SNN, überwacht die Kommunikation und greift bei Bedarf ein. Dieser Kontroll-Schicht ist die Dev-Ops Schicht (blau) mit dem Management-System überstellt. Durch diese Schicht können Eingriffe, wie beispielsweise Updates, am Straßenzug vorgenommen werden.

Abbildung 2.3 zeigt eine mögliche Ausprägung eines Micro-Grids, bestehend aus drei Nano-Grids (grün). In der Abbildung wird das Micro-Grid als smarter Straßenzug die Nano-Grids in Form von smarten Wohnhäusern repräsentiert. Die Kontroll-Schicht(orange) ist mit allen Wohnhäusern verbunden. Über ihr liegt die DevOps-Schicht (blau). Ein typisches Szenario auf dieser Ebene ist eine gegenseitige Versorgung der Wohnhäuser. Angenommen das linke Haus besitzt eine PVA, welche aufgrund einstrahlenden Sonnenlichts Strom produziert. In dem Haus sind nur wenige Verbraucher an, weil beispielsweise die Bewohner momentan im Urlaub sind. Durch die wenigen Verbraucher und die bereits aufgeladenen Batterien wird der insgesamt produzierte Strom nicht innerhalb des Hauses verbraucht. Nach der obigen Beschreibung des Nano-Grids würde in dieser Situation der Hausanschluss reagieren und der überschüssige Strom zum aktuellen Strompreis in das öffentliche Netz ein-

gespeist werden. Da die einzelnen Häuser mittels ihrer Energiemanager miteinander kommunizieren können, greift der Energiemanager ein. Dieser hat soeben von seinem benachbarten Energiemanager (mittleres Haus) erfahren, dass ein Stromdefizit vorliegt und bietet den überschüssigen Strom innerhalb der Nachbarschaft an. So wird Strom in dem Stadtteil lokal produziert und auch in diesem verbraucht.

2.1.3. Urban-Grid

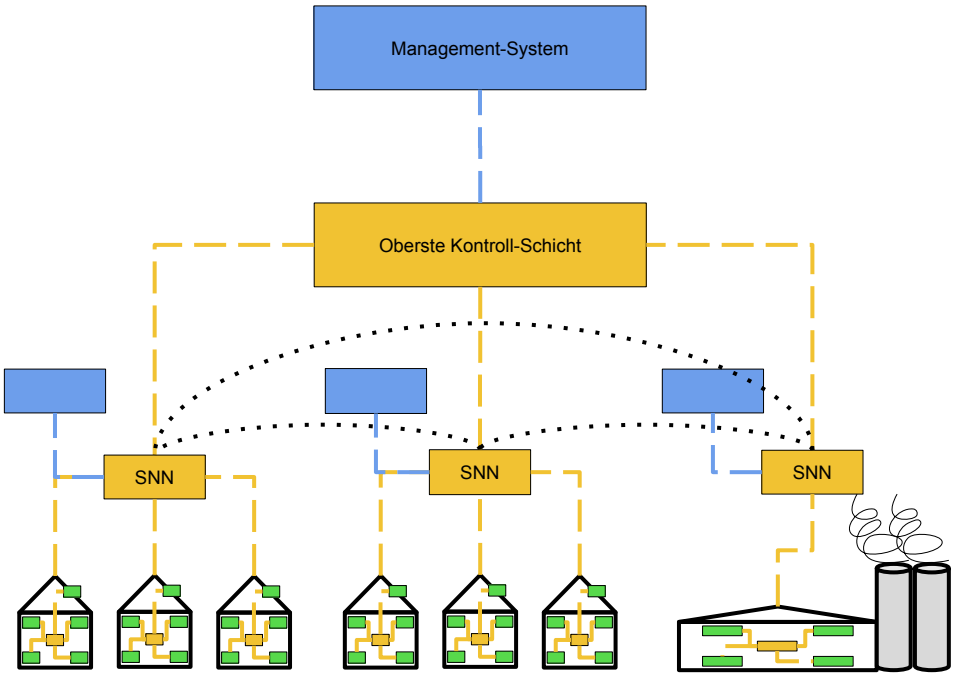


Abbildung 2.4.: Urban-Grid: Stromversorgung einer Stadt. Die abgebildete Stadt besteht aus drei Micro-Grids, dargestellt durch zwei Straßenzüge und ein Kraftwerk. Die Micro-Grids können über die SNN direkt miteinander interagieren. Die oberste Kontroll-Schicht (orange) überwacht die Kommunikation der SNN und kann in die Handlungen eingreifen. Dieser Schicht ist eine DevOps-Schicht (blau) übergeordnet. Von der DevOps-Schicht aus können systemweite Veränderungen eingespielt werden.

Der Zusammenschluss mehrerer Micro-Grids wird als Urban-Grid bezeichnet. Ein Urban-Grid bildet eine smarte Stadt (*Smart-City*) ab. Das Ziel bei der Zusammenfassung der Micro-Grids zu einer Smart City ist zum einen die gruppenübergreifende Kommunikation der Stadtteile bzw. Straßenzüge und zum anderen die Überwachung des Gesamtsystems. Durch den Zusammenschluss der Micro-Grids können systemweite Kontrolle ausgeübt und Änderungen angestoßen werden.

Abbildung 2.4 auf der vorherigen Seite visualisiert ein Urban-Grid, bestehend aus drei Micro-Grids. Über den Micro-Grids ist eine weitere Kontroll-Schicht angeordnet. Diese Schicht dient der Kommunikationskontrolle der untergeordneten SNNs. Die SNNs der Micro-Grids können dabei direkt miteinander interagieren. Das übergeordnete Management-System ist die oberste Instanz. Von dieser Schicht aus können systemweite Änderungen, wie Updates und Verbrauchsmuster, in das System eingespielt werden. Des Weiteren sind in der Abbildung zwei smarte Straßenzüge und ein Kraftwerk zu sehen. Mögliche Szenarien sind zum einen die Über-/Unterversorgung eines Stadtteils und daraus resultierende gegenseitige Versorgung und Abnahme. Auf dieses Szenario wird nicht weiter eingegangen, da es analog zu einem Micro-Grid funktioniert, mit der Ausnahme, dass die SNNs miteinander verhandeln und die übergeordnete Kontroll-Schicht diese Kommunikation überwacht. Ein weiteres mögliches Szenario der externe Eingriff in das laufende System. Wie schon im Fall eines Micro-Grids können Updates und Änderungen eingebracht werden. In diesem Beispiel wird ein weiterer Stromerzeuger in das System aufgenommen. Dies kann beispielsweise ein Solarfeld oder eine Windkraftanlage sein. Die notwendige Software für den Betrieb des Erzeugers, sowie Regeln, wann die Kontroll-Schichten in das Verhalten des Erzeugers eingreifen sollen, werden von einem Entwicklerteam erstellt. Nach der physischen Fertigstellung des Erzeugers können die neuen Regeln in das System eingebracht werden. Dazu werden die betroffenen Kontroll-Schichten informiert und erhalten die aktualisierten Regeln. Nach ein paar Monaten Betrieb steht ein Softwareupdate für den neuen Erzeuger bereit. Über den zugehörigen SNN wird der Energiemanager des Erzeugers kontaktiert. Der Energiemanager lädt das Update herunter und reicht es an den untergeordneten Erzeuger weiter.

2.1.4. Zusammenfassung der Szenarien

Die drei Szenarien haben gemeinsam, dass sie den Entwurf und den Betrieb eines komplexen, selbstorganisierenden Software-Systems beschreiben. Innerhalb dieser komplexen Systeme existieren viele kleine Systeme, in Form von einzelnen IoT-Devices oder Zusammenschlüssen zu smarten Häusern und Straßenzügen, welche miteinander kommunizieren, interagieren und so gemeinsame Absprachen treffen. Durch die Interaktionen der kleinen (Teil-)Systeme kann innerhalb des Gesamtsystems unerwartetes, neues Verhalten auftreten. Dieses Verhalten wird als Emergenz bezeichnet. In dem nachfolgenden Abschnitt wird der Begriff der Emergenz und die zugrundeliegende Bedeutung ausführlich beschrieben. Weiterhin wird in dem Abschnitt geklärt, wodurch sich selbstorganisierende Systeme auszeichnen und welche spezifischen Eigenschaften diese Systeme haben müssen.

2.2. Selbstorganisation und Emergenz in (IoT-)Systemen

Dieses Kapitel beschreibt zwei wichtige Konzepte im Zusammenhang mit komplexen Systemen. Die beiden Konzepte heißen *Selbstorganisation* und *Emergenz* bzw. *emergentes Verhalten*. Ein selbstorganisierendes System kann ohne externen Eingriff auf Veränderungen reagieren und interne Anpassungen durchführen. Die Anpassungen können Effekte auf das Gesamtsystem nach sich ziehen. Emergentes Verhalten kann auftreten, wenn innerhalb eines Systems verschiedene Komponenten eigenständig miteinander interagieren. Die Auswirkungen des neuen Verhaltens auf das laufende System fallen dabei von System zu System unterschiedlich aus und können somit einen positiven oder einen negativen Effekt mit sich bringen. Neben der konkreten Definition des Begriffs wird auf die folgenden Punkte eingegangen: (1) Auswirkungen von Emergenz und (2) Emergenz erkennen.

2.2.1. Selbstorganisation

Ein selbstorganisierendes System wird in dieser Masterarbeit, angelehnt an [MST17, S. 72] wie folgt definiert:

Definition 2.1 (Selbstorganisierendes System)**.** Wenn einzelne Entitäten oder Gruppen von interagierenden Entitäten ihre Struktur oder ihr Verhalten ohne den Einfluss einer Kontrollinstanz verändern können, spricht man von einem selbstorganisierenden System. Der Begriff Selbstorganisation ist verwandt mit Autonomie.

Die Definition von Selbstorganisation beinhaltet, dass das betrachtete System nicht statisch, sondern dynamisch ist. Das ist dadurch gekennzeichnet, dass im Verlauf Änderungen auftreten und sich das System mit der Zeit entwickelt. Die Entwicklungsprozesse werden dabei intern und folglich dezentral durch die autonomen Entitäten entschieden und ausgeführt [MST17, S. 14]. Für die Änderungen bedarf keiner zentralen, externen Verwaltung. Insbesondere bei der Betrachtung verteilter Systeme, zu denen auch IoT-Systeme zählen, ist der Verzicht einer zentralen (Kontroll-)Instanz wünschenswert. Eine zentrale Instanz, die jegliche Funktionalität und Entscheidungsmacht beinhaltet, birgt in einem solchen System die Gefahr eines *Bottlenecks*, wenn in dem Fall einer Anpassung die Änderungen von einer einzelnen Instanz ausgehen und alle weiteren Instanzen den zeitgleichen Zugriff anstreben [TS02, S. 10]. Zentralität wirkt sich zudem negativ auf die Zuverlässigkeit eines Systems aus [LSL06]. Das Prinzip der Verteilung der Funktionalitäten ist eine wichtige Methode, um einerseits Skalierbarkeit zu erreichen [TS02, S. 13] und andererseits eine gewisse Modularität und Flexibilität zu erhalten [LSL06].

Drei wichtige Charakteristika eines selbstorganisierenden Systems sind (1) die Zunahme der Ordnung, (2) die Autonomie und (3) die Anpassbarkeit und Robustheit [WH04]. Das erste Charakteristikum, die Zunahme der Ordnung, gibt dabei an, dass in dem System bereits eine gewisse initiale Ordnung besteht. In natürlichen und auch technischen Systemen ist in der Regel eine bestimmte Ordnung vorhanden [MST17, S. 26]. Ohne eine bestehende Ordnung kann ein System kein sinnvolles Verhalten aufweisen. Ist der Grad der Ordnung hingegen zu hoch, resultiert dies in einer zu hohen Komplexität des Systems. Infolgedessen muss das selbstorganisierende System eine Balance in dem Grad der Ordnung finden [WH04]. Die Autonomie manifestiert die eigenständige Organisation und Anpassung des Systems ohne den Einfluss externer Kontrollinstanzen. Die Anpassbarkeit und Robustheit beinhalten, dass das System auf Änderungen reagieren kann und Organisation stattfindet [WH04]. Selbstorganisation kann somit eine gewisse Stabilität innerhalb eines Systems bewirken.

Selbstorganisation ist heutzutage unter anderem in natürlichen, sozialen und technischen Systemen anzutreffen. Die nachfolgenden Beispiele für diese Systeme sind an [MST17, S. 27 ff.] angelehnt. Eine Form eines natürlichen, selbstorganisierenden Systems ist eine simple Ameisenkolonie. In dieser Kolonie interagieren die individuellen Ameisen miteinander und tauschen so Informationen über Futterquellen aus. Durch die Interaktionen entsteht emergentes Verhalten, weshalb auf dieses Beispiel in Unterabschnitt 2.2.2 auf der nächsten Seite näher eingegangen wird. Als Beispiel für ein soziales und technisches System wird die Verkehrsregulierung betrachtet. Auf der einen Seite kann ein Verkehrssystem als Interaktion von Autos, Fahrrädern, Fußgängern und weiteren betrachtet werden. In einem solchen System gelten Regeln wie *Rechts vor Links* oder das Beachten von Schildern und Lichtsignalen. Fällt beispielsweise die Lichtanlage einer Kreuzung aus, so können die Verkehrsteilnehmer auf diese Änderungen selbstständig reagieren und ihr Verhalten entsprechend anpassen. Autofahrer fahren mit Vorsicht an die Kreuzung heran und Fußgänger überqueren nicht einfach die Straße. Durch die Anpassungen kommt es in der Regel nicht zu einem Systemzusammenbruch. Auf der anderen Seite kann ein modernes Verkehrssystem als ein Netzwerk aus Sensoren und Aktuatoren angesehen werden. Die Sensoren können beispielsweise ein erhöhtes Verkehrsaufkommen erfassen. Auf Basis dieser Information können unter anderem zulässige Geschwindigkeiten oder Ampelschaltungen angepasst werden.

Selbstorganisation innerhalb eines Systems bringt eine dezentrale Verwaltung mit sich. Dadurch kann Skalierbarkeit und Stabilität erreicht werden. Durch die Interaktionen innerhalb des Gesamtsystem und die Anpassungen kann jedoch neues, unerwartetes Verhalten, sogenannte *Emergenz* entstehen.

2.2.2. Emergenz

Im Folgenden wird der Begriff Emergenz definiert. Die Definition ist angelehnt an [MR15] und [MST17, S. 72].

Definition 2.2 (Emergenz)**.** Der Begriff *Emergenz* hat seinen Ursprung in dem lateinischen Wort *emegere*, was soviel bedeutet wie *bekannt werden* oder *ans Licht kommen*. Emergenz ist ein Verhalten des Gesamtsystems, welches vorab nicht explizit programmiert wurde und durch die Interaktionen der einzelnen Komponenten entsteht. Emergentes Verhalten ist nicht bei einer isolierten Komponente beobachtbar. Das Auftreten von Emergenz ist in der Regel nicht vorhersehbar.

Wie in der Definition ersichtlich wird, ist emergentes Verhalten ein neues Verhalten, welches durch Interaktionen von Teilsystemen oder Komponenten entsteht und sich erst auf höherer Ebene bemerkbar macht. Das Verhalten tritt nicht bei isolierten Teilen auf und kann zu keiner individuellen Komponente zurückverfolgt werden. Es lässt sich nicht durch das explizit programmierte Verhalten erklären [MR15]. Die Komplexität des Agentenverhaltens spielt bei der Entstehung von Emergenz eine eher untergeordnete Rolle. Selbst sehr einfache Regeln und Verhaltensweisen der Agenten können in komplexen Gesamtsystemen resultieren [WR17]. Nach Weyer und Roos [WR17] resultiert emergentes Verhalten aus dem Wechselspiel zwischen Komponenten- und System-Ebene. Damit lässt sich die Entstehung von Emergenz grundsätzlich auf Interaktionen zurückführen.

Emergentes Verhalten kann in vier Kategorien unterteilt werden: (1) Simpel, (2) Schwach, (3) Stark und (4) Unheimlich [MR15]. Abbildung 2.5 auf der nächsten Seite stellt diese Unterteilung schematisch dar. Die simple Emergenz kann dabei durch einfache Modelle vorhergesagt werden. Schwache Emergenz kann durch ein vereinfachtes Modell der Realität erklärt und in Simulationen reproduziert werden, aber nicht mehr solide vorhergesagt werden. Starke Emergenz kann nicht mehr in einem vereinfachten Modell reproduziert werden. Eine Simulation des Systems deckt eventuell die Emergenz auf. Unheimliche Emergenz wird in keinem Modell des Systems reproduziert, auch nicht in einem Modell mit der Komplexität des realen Systems [MR15]. Während sich simple und schwache Emergenz in einem deterministischen Raum bewegen, kann die starke Emergenz nur noch über Heuristiken ergründet werden. Die unheimliche Emergenz bewegt sich in einem unbegrenzten Suchraum.

Die Auswirkungen von Emergenz auf ein System können ganz unterschiedlich sein. Es können sich folglich positive oder negative Effekte für das Gesamtsystem ergeben. Ein Beispiel für eine negative Auswirkung ist ein klassischer Verkehrsstau [WR17]. Ein Auto kann als unabhängige Instanz mit relativ simplem Verhalten beschrieben werden. Das Auto bewegt sich zusammen mit weiteren Fahrzeugen in dem System

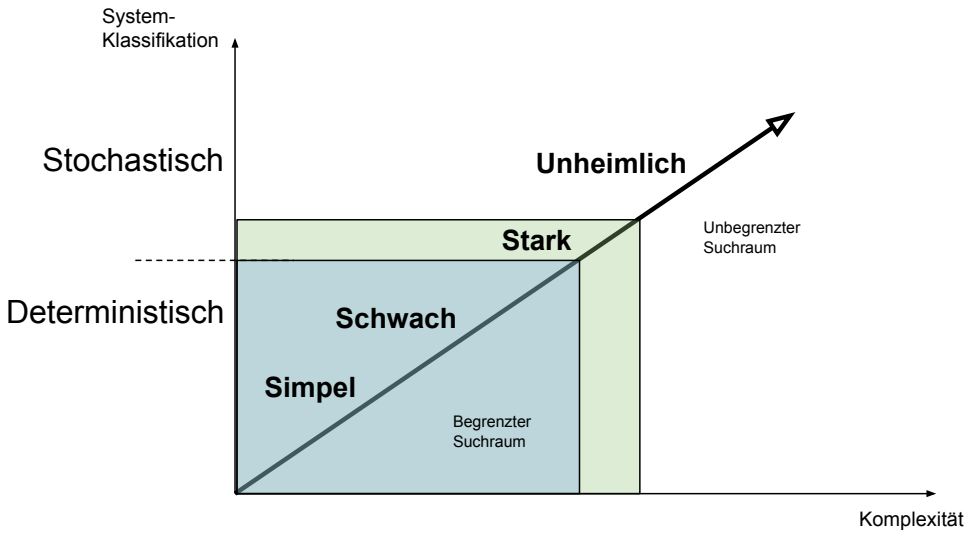


Abbildung 2.5.: Kategorien von Emergenz. Abgebildet sind die vier Katergorien von emergentem Verhalten: (1) Simpel, (2) Schwach, (3) Stark und (4) Unheimlich. Simple Emergenz charakterisiert, dass sie durch einfache Modelle vorhergesagt werden kann. Schwache Emergenz kann zwar durch ein vereinfachtes Modell der Realität erklärt, aber nicht mehr solide vorhergesagt werden. Starke Emergenz kann nicht mehr in einem vereinfachten Modell reproduziert werden. Eine Simulation des System deckt eventuell die Emergenz auf. Unheimliche Emergenz wird in keinem Modell des Systems reproduziert. Die Grafik ist entnommen aus [MR15] und wurde ins Deutsche übersetzt.

Verkehr. Durch gegenseitige Rücksichtnahme und hintereinander Fahren interagieren die Autos miteinander. Bremst nun beispielsweise ein Auto abrupt ab, müssen die nachfolgenden Autos ebenfalls reagieren. Als Folge davon kann ein Stau oder gar Unfall entstehen. Dieses neue Verhalten ist nicht explizit beabsichtigt und kann nicht bei einem isolierten Auto beobachtet werden.

Eine positive Auswirkung von Emergenz ist in [MST17, S. 38 f.] beschrieben. Das Beispiel handelt von der Futtersuche einer Ameisenkolonie. Die Ameisen scheren in alle Richtungen aus. Findet eine Ameise eine Futterquelle, hinterlegen die Ameisen Pheromone auf ihrem Rückweg zum Nest. Je höher die Konzentration an Pheromonen, desto bestrebter sind nachfolgende Ameisen diesen Weg zu nehmen. Damit verlagern sich die Laufrichtungen der Ameisen auf einige wenige Routen. Die Interaktionen der einzelnen Ameisen führen so zu neuem Verhalten innerhalb des Gesamtsystems, welches wieder keiner einzelnen Komponente zuzuordnen ist.

Die Beispiele zeigen, dass Emergenz nicht pauschal als positiv oder negativ angesehen werden kann. Aus diesem Grund sollte nicht das Ziel bei der Entwicklung eines Systems sein, emergentes Verhalten zu unterdrücken oder zu vermeiden. Eine Ver-

meidung kann nach [LSL06] erreicht werden, indem unter anderem Nichtlinearitäten aufgedeckt und entfernt und zentrale Algorithmen verwendet werden. Vielmehr ist es wichtig, dass vor dem Betrieb des System mögliche emergente Verhaltensweisen erkannt und untersucht werden. In dem Projekt *DevOpt* werden zudem Ansätze diskutiert und erforscht, die ein kontrolliertes Auftreten der Emergenzen und deren Auswirkungen zum Ziel haben.

Nach der Definition von Emergenz zu Beginn dieses Unterabschnitts tritt emergentes Verhalten erst im laufenden Betrieb des Systems auf. Somit ist es prinzipiell nicht möglich, alle Auswirkungen des neuen Verhaltens vorab zu bestimmen. Bereits einfache Regeln der einzelnen Komponenten können dabei zu einem sehr komplexen Verhalten innerhalb des Gesamtsystems führen [MN08]. Wie bereits bei der Kategorisierung von Emergenz angesprochen, stellt die Simulation des Systems eine effektive Methode für die Erkennung emergenten Verhaltens dar [MR15]. Durch die Variation der Modellparameter während der Simulation kann zudem die Reaktion des Systems auf Änderungen beobachtet werden [WR17]. Damit kann emergentes Verhalten sichtbar werden, im Fall der starken oder unheimlichen Emergenz ist dies jedoch nicht garantiert [MR15].

Die Konzepte *Selbstorgansiation* und *Emergenz* sind, wie die Definitionen und Erklärungen zeigen, sehr eng miteinander verwandt. Diese Beziehung äußert sich unter anderem darin, dass Emergenz durch Selbstorganisation entstehen kann. Komplexe (IoT-)Systeme, wie sie in dieser Masterarbeit innerhalb der drei vorgestellten Szenarien beschrieben wurden, sind selbstorganisierend und erfüllen die Voraussetzungen für die Entstehung emergenten Verhaltens. Um diese Systeme adäquat zu modellieren und anschließend simulieren zu können, verhandelt der nachfolgende Abschnitt eine geeignete Methode zur Modellbildung.

2.3. Agentenbasierte Modellierung und Simulation

Die Agentenbasierte Modellierung und Simulation (ABMS) ist ein Ansatz für die Modellierung eines komplexen Systems aus interagierenden Agenten [MN10]. Die ABMS soll die Dynamik des Systems erfassen, die durch die Agenteninteraktionen entsteht [MN10] und die Auswirkungen des Handelns abschätzen. Wie in Unterabschnitt 2.2.2 auf Seite 14 beschrieben, reicht bereits simples Agentenverhalten aus, um ein komplexes Gesamtsystem zu erhalten. Aufgrund ihrer Flexibilität stellen Macal und North [MN06] die Bedeutung der ABMS gleich mit der Erfindung relationaler Datenbanken. In diesem Abschnitt werden zu Beginn Anwendungsfelder für die ABMS vorgestellt. Im Anschluss wird die Methode erläutert und durchlaufen. Das Ende des Abschnitts bildet eine Übersicht über die Vorteile der ABMS und stellt alternative Ansätze zur Modellierung komplexer Systeme vor.

Das wohl wichtigste Instrument im Rahmen der ABMS ist ein Agent. Die Definition eines simplen Agenten ist angelehnt an [RN03, S. 32] und [VRH04, S. 350]. Ein Agent wird wie folgt definiert:

Definition 2.3 (Agent)**.** Ein Agent ist eine Entität, die handelt. Der Wortursprung kommt aus dem lateinischen Wort *agere* und bedeutet so viel wie *machen*. Ein Agent nimmt seine Umgebung über Sensoren wahr und handelt mit seinen Aktuatoren. Agenten handeln autonom und unabhängig voneinander. Mit einem Agenten können Nachrichten ausgetauscht werden.

Für die Zwecke der Masterarbeit muss ein Agent die folgenden essentiellen Eingenschaften besitzen ([MN06] und [MN10]):

- Ein Agent ist eine modulare und identifizierbare Instanz mit einer Menge von Eigenschaften (Attributen) und Verhaltensregeln. Die Regeln determinieren die Entscheidungsfähigkeit. Die Modularität erzwingt eine Grenze des Agenten, sodass eindeutig ist, was zum Agenten gehört und was nicht.
- Ein Agent ist autonom und kann unabhängig von anderen Agenten handeln.
- Ein Agent hat einen Zustand, welcher sich über die Zeit ändern kann.
- Ein Agent ist sozial, indem er mit anderen Agenten interagiert. Die Interaktionen finden über Protokolle statt. Ein Agent ist dabei reaktiv und proaktiv [TS02, S. 175]. Eine Art der Interaktion ist die Kommunikation.
- Ein Agent lebt in einer Welt zusammen mit anderen Agenten und interagiert mit seiner Umgebung.

Fundamental ist dabei die Eigenschaft der Autonomie, da sie dem Agenten eine gewisse Freiheit ermöglicht und dieser somit unabhängig von anderen Agenten handeln kann. Weiterhin kann ein Agent nach Macal und North [MN10] die folgenden optionalen Eigenschaften besitzen:

- Ein Agent kann flexibel sein und die Fähigkeit haben, sein Verhalten anzupassen und zu lernen.
- Ein Agent kann zielorientiert sein.
- Ein Agent kann mobil sein und sich in seiner Umgebung bewegen ([TS02, S. 175]).
- Agenten einer Welt können heterogen sein.

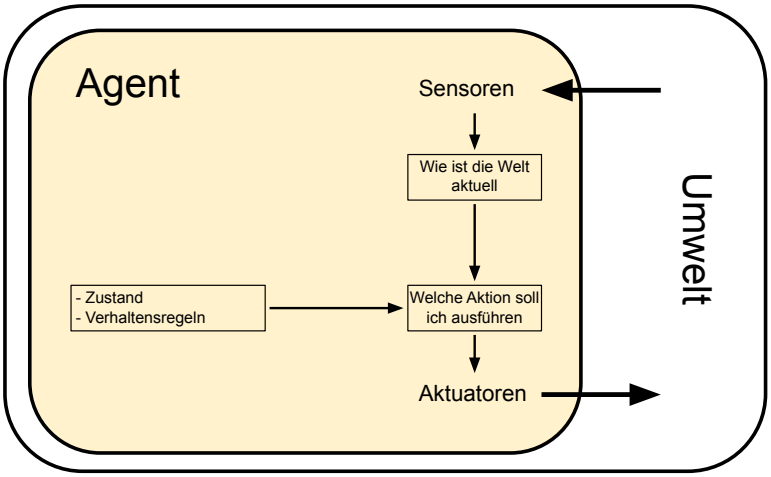


Abbildung 2.6.: Schematische Darstellung eines Agenten. Die Grafik ist angelehnt an [RN03, S. 47] und [MN08]. Ein Agent nimmt seine Umwelt und andere Agenten über Sensoren wahr. Die Wahrnehmung kann dabei auch über einen Nachrichtenempfang stattfinden. Auf Grundlage der aktuellen Umgebung, dem Zustand des Agenten und der eigenen Verhaltensregeln wählt der Agent eine Aktion aus. Über seine Aktuatoren gelangen die Auswirkungen der Aktion an die Umgebung und die übrigen Agenten.

In Abbildung 2.6 ist schematisch ein simpler Agent dargestellt. Wie in der Abbildung zu erkennen ist, hat der Agent eine definierte Grenze. Die Grenze trennt die ihn umgebene Umwelt von seinem internen Zustand und den Verhaltensregeln. Interaktionen nach außen finden über spezifizierte Schnittstellen, die Sensoren und Aktuatoren, statt. Ein so vorhandener Agent kann nach Festlegung der Regeln in verschiedenen Szenarien eingesetzt werden. In dem nachfoldenden Unterabschnitt ist eine Übersicht über mögliche Anwendungsbereiche von Agenten und der ABMS zur Modellierung gegeben.

2.3.1. Anwendungen für die ABMS

Die agentenbasierte Modellierung ist eine sehr flexible Methode, die in nahezu allen Bereichen der Forschung und Wirtschaft Anwendung findet. In Tabelle 2.1 auf der nächsten Seite ist eine Übersicht über einige dieser Anwendungsbereiche gegeben. Die dort abgebildeten Anwendungsbereiche bewegen sich dabei in einem breiten Sprektrum, welches die Natur-, Sozial- und Wirtschaftswissenschaften einschließt. Die vielfältigen Anwendungsbeispiele haben gemeinsam, dass Individuen identifizierbar sind und diese Individuen in einem Austausch miteinander und mit ihrer

Unternehmen und Organisationen	**Gesellschaft und Kultur**
• Fertigungsbetriebe	• Antike Zivilisationen
• Lieferketten	• Soziale Faktoren des Terrorismus
• Verbrauchermärkte	• Organisatorische Netzwerke
• Versicherungsbranche	
Wirtschaft	**Menschenansammlungen**
• Künstliche Finanzmärkte	• Fußgängerbewegungen
• Handelsnetze (auch Energiewirtschaft)	• Evakuierungsmodellierung
Infrastruktur	**Biologie**
• Transportwesen und Verkehr	• Populationsdynamiken
• Strommärkte	• Ökologische Netzwerke
• Wasserstoff-Infrastruktur	• Verhalten von Tiergruppen/ Herden
	• Zellverhalten und Zellprozesse

Tabelle 2.1.: Anwendungsgebiete für die ABMS. Die Tabelle ist angelehnt an [MN08] und illustriert die Flexibilität der ABMS für die Modellierung komplexer Systeme. Komplexe Systeme treten heutzutage in vielen Bereichen der Forschung und Wirtschaft auf, in denen künstliche oder reelle Individuen identifizierbar sind und im Austausch miteinander stehen.

Umwelt stehen. Daher kann mittels der ABMS in der Biologie das Herdenverhalten von Tieren modelliert und simuliert werden. Jedes individuelle Tier einer Herde wird dabei als separater Agent mit Eigenschaften und Verhaltensweisen beschrieben. Betrachtet man Lieferketten in Unternehmen, so ist jedes Glied dieser Kette identifizierbar und es kann das Zusammenwirken der einzelnen Glieder beobachtet werden. In Handelsnetzen in der Wirtschaft treten die Handelspartner als unabhängige und autonome Individuen auf.

Zusammenfassend lässt sich festhalten, dass der Ansatz dann Anwendung findet, wenn Entitäten eines Szenarios eindeutig identifizierbar sind und als Agenten repräsentiert werden können. Außerdem müssen diese Entitäten in gegenseitigem Austausch stehen. Nachdem in diesem Abschnitt die Flexibilität des Ansatzes herausgestellt wurde, wird nachfolgend die Methode ausführlich erläutert.

2.3.2. Die Methode ABMS

Ein agentenbasiertes Modell besteht nach Macal und North [MN10] fundamental aus drei Elementen. Diese sind (1) eine Menge von Agenten mit Attributen und Verhaltensregeln, (2) eine Menge von Beziehungen zwischen den Agenten, welche

definieren wie und mit wem die Agenten interagieren dürfen und (3) die Umwelt, mit welcher die Agenten interagieren können. Somit besteht die Aufgabe bei der Entwicklung eines agentenbasierten Modells in der Definition der Agenten und der Umwelt, der Bestimmung der Agentenmethoden und -interaktionen sowie der finalen Implementierung [MN06]. Als Hilfestellung für die Entwicklung eines agentenbasierten Modells empfehlen Macal und North [MN10] die Beantwortung der sieben aufgeführten Kernfragen:

1. Welches spezifische Problem soll durch das Modell gelöst werden? Welche spezifische Frage soll das Modell klären? Welchen Mehrwert liefert die agentenbasierte Modellierung?
2. Was sind die Agenten in dem Modell? Wer sind die Entscheidungsträger in dem System? Welches sind die Entitäten, die Verhaltensweisen haben? Welche Attribute sind beschreibend (statische Attribute)? Welche Attribute werden durch das Modell berechnet und angepasst (dynamische Attribute)?
3. Wie sieht die Umwelt der Agenten aus? Wie interagieren die Agenten mit der Umwelt? Spielt die Bewegung eines Agenten in der Umwelt eine wichtige Rolle?
4. Welches Agentenverhalten ist von Interesse? Welche Entscheidungen treffen die Agenten? Nach welchen Verhaltensweisen wird gehandelt? Welche Aktionen werden unternommen?
5. Wie interagieren die Agenten miteinander? Wie interagieren die Agenten mit ihrer Umwelt? Wie expansiv oder fokussiert sind die Interaktionen der Agenten (Ausmaß der Interaktionen)?
6. Woher stammen die Daten für das Modell und insbesondere über das Verhalten der Agenten?
7. Wie kann das gesamte Modell und insbesondere das Verhalten der Agenten validiert werden?

Die ausführliche Beantwortung der sieben Kernfragen stellt einen wesentlichen Teil der Entwicklung eines agentenbasierten Modells dar. Für die Entwicklung eines solchen Systems bietet sich eine bottom-up Vorgehensweise an. Bei diesem Vorgehen wird das Modell von der Agenten-Ebene ausgehend iterativ bis zu der Systemebene erstellt [MN10]. Dieser Vorgehensweise liegt die Annahme zugrunde, dass die Mechanismen, welche die Systemdynamik erklären, nicht auf der Systemebene sondern auf der Agenten-Ebene zu finden sind [WR17]. Dieser aufbauende Ansatz beinhaltet jedoch ein Problem. Angenommen die modellierten Agenten erzeugen während der Simulation ein reales Verhalten, ist damit nicht bewiesen, dass die Agenten tatsächlich die Ursache für das reale Verhalten sind. „Es wird damit eine mögliche

Erklärung geliefert, die unter Umständen nur eine von vielen ist.“[WR17]. Nichtsdestotrotz „[...] besteht ihr Wert darin, die Dynamik komplexer soziotechnischer Systeme untersuchbar zu machen [...]“[WR17] und damit potenzielles emergentes Verhalten darzustellen [MR15].

2.3.3. Kritik an ABMS

Wie in den vorherigen Abschnitten bereits herausgestellt, ist ABMS eine sehr flexible Methode für die Modellierung agentenbasierter Systeme. Aufgrund ihrer Beliebtheit steht eine Vielzahl an Tools bereit. Diese bewegen sich in einem Spektrum von kleinen Desktopanwendungen, beispielsweise mit VBA, MATLAB oder NetLogo, über Entwicklungsumgebungen (z.B. Repast oder AnyLogic) bis hin zu mächtigen Frameworks als Unterstützung für die gängigen Programmiersprachen wie Python oder Java ([MN08], [MSL+19] and [BCPR03]).

Ein Kritikpunkt nach [WR17] stellt die Validierung des entwickelten Modells dar. Damit ein Modell einen realen Zustand nachahmen kann, bedarf es an geeigneten Modellparametern. Die Erhebung dieser Parameter über große Datenmengen stellt oftmals eine große Herausforderung dar. Weyer und Roos argumentieren dagegen, dass die ABMS trotz dieses Problems sehr gut für die Folgenabschätzung geeigent ist, da die benötigten Parameter variiert werden können und so eine Betrachtung der Systemreaktion auf Änderungen ermöglicht [WR17].

Weiterhin wird der ABMS zum einen ein zu hoher Grad Abstraktion und damit Detailverlust und zum anderen eine zu große Komplexität im Vergleich zu mathematischen Modellen vorgeworfen [WR17]. Nach Weyer und Roos „[...] trifft gerade [die ABMS] das richtige Maß an Detailreichtum und Abstraktheit und kann damit ein Bindeglied zwischen mathematischen und nichtmathematischen Forschungsansätzen sein.“[WR17]. Weiterhin „[...] trägt [die ABMS] damit zu einem vertieften Verständnis komplexer Systeme bei.“[WR17], weshalb diese Methode schlussendlich in dieser Arbeit bei der Konzeptentwicklung zum Einsatz kommt.

Das Kapitel hat zu Beginn Szenarien eines Electric-Grids beschrieben und erklärt, was man unter einem Nano-, Micro- und Urban-Grid versteht. Die Grids sind Beispiele für selbstorganisierende Systeme, die während des Betriebs neues Verhalten entwickeln können. Im weiteren Verlauf wurden die Begriffe *Selbstorganisation* und *Emergenz* definiert und beschrieben. Um mögliche Emergenzen vor der Inbetriebnahme zu erfassen, hat sich in der Literatur die Simulation bewährt. Mit der ABMS wurde eine geeignete Methode zur Entwicklung eines Modell präsentiert. Mittels dieser Grundlagen unter Einsatz der Methode wurde ein Simulator-Konzept entwickelt. Der Entwicklungsprozess wird im nachfolgenden Kapitel detailliert beschrieben.

3. Konzeption des Simulators

Dieses Kapitel bildet den theoretischen Kern dieser Masterarbeit. Wie zu Beginn der Arbeit herausgestellt wurde, besteht das Ziel der Arbeit in der Konzeptentwicklung eines Szenarien-Simulators für selbst-optimierende, emergente Syteme. In dem Kapitel wird die Entwicklungshistorie des Konzepts durchlaufen und dabei auf wichtige Designentscheidungen Bezug genommen. Am Ende des Kapitel wird das fertige Konzept noch einmal resümiert.

Das Kapitel 3 gliedert sich in fünf Abschnitte. Der erste Abschnitt nimmt Bezug auf die in Abschnitt 2.1 auf Seite 5 beschriebenen Szenarien und analysiert die Szenarien in Hinblick auf Gemeinsamkeiten und Unterschiede. Im Anschluss an die Analyse werden in dem zweiten Abschnitt die genauen Anforderungen an den Simulator definiert. Insgesamt existieren 14 Anforderungen, die sich in zwei Kategorien, funktionale und qualitative Anforderungen, gliedern. In dem dritten Abschnitt wird die in Abschnitt 2.3 auf Seite 16 erklärte ABMS angewendet. Mithilfe der Methode werden die relevanten Aspekte des Konzepts, genauer die Agenten, die Agentenbeziehungen und -interaktionen und die Umwelt, spezifiziert. Anknüpfend an das agentenbasierte Modell erläutert der vierte Abschnitt detailliert das entwickelte Kommunikationsprotokoll. Der letzte Abschnitt stellt die Architektur des Simulator-Konzepts vor und wägt dabei alternative Architekturen begründet ab. Die Architektur bildet das Zusammenspiel und die Abhängigkeiten der Module des Simulators ab. Das Konzept wird somit iterativ über die Abschnitte erstellt.

3.1. Analyse der Projektszenarien

In diesem Abschnitt werden die drei in Abschnitt 2.1 auf Seite 5 vorgestellten Szenarien genauer analysiert. Die Analyse stellt dabei das Nano-Grid, das Micro-Grid und das Urban-Grid gegenüber, um Überlappungen hinsichtlich des Aufbaus und Ablaufs festzustellen. Weiterhin wird durch die Analyse geklärt, an welchen Stellen individuelle Steuerungsmechanismen und Vorgehensweisen auftreten. Somit liegt der Fokus auf den Gemeinsamkeiten und Unterschieden. Eine abschließende Übersicht fasst die wesentlichen Aspekte des Analyseergebnisses kurz zusammen. In dieser

Übersicht werden auch die für die Entwicklung des softwarebasierten Simulators relevanten Aspekte herausgestellt. Das Ergebnis der Szenarien-Analyse fließt direkt in die angrenzende Anforderungsdefinition ein.

3.1.1. Gemeinsamkeiten

Eine der größten Gemeinsamkeiten ist, dass die Szenarien ein komplexes, selbstorganisierendes IoT-System beschreiben, in dem während des Betriebs neues, unbeabsichtigtes Verhalten auftreten kann. Dieses emergente Verhalten ist dabei bei keinem der IoT-Devices isoliert zu beobachten, sondern entsteht durch die vielfältigen Interaktionen mit anderen Agenten und mit der Umwelt. Damit ist eine Gemeinsamkeit, dass in den drei Szenarien jeweils smarte Komponenten zum Einsatz kommmen, welche die Stromflüsse autonom verhandeln. Die smarten Komponenten kommunizieren dazu miteinander und haben somit Überlappungen bezüglich ihres Verhaltens (genaueres in Abschnitt 3.3 auf Seite 34). Insbesondere die gemeinsame Eigenschaft der Autonomie, der Interaktionen untereinander und die modulare Identifizierbarkeit sind an dieser Stelle hervorzuheben. Diese Eigenschaften machen nach Definition 2.3 auf Seite 17 die smarten Komponenten zu Agenten. Die Interaktionen der Agenten stellen dabei eine weitere und die wohl wichtigste Gemeinsamkeit dar. Hinzu kommt der jeweils hierarchische Aufbau der Grids aus kleinen Systemen. Innerhalb der kleinen Systeme finden Gruppierungen statt. Im Fall eines Nano-Grids bilden die smarten Devices, beispielsweise eines Wohnhauses, eine Gruppe. Diese Gruppe kann mit weiteren Gruppen von IoT-Devices verknüpft werden. Es entsteht ein Micro-Grid. Diese können wiederum zu einem Urban-Grid kombiniert werden. Jedes Grid ist somit hierarchisch aufgebaut und beinhaltet IoT-Devices auf der untersten Ebene. Ein Grid kann somit als *System-of-Systems* beschrieben werden [MST17, S. 214/S. 222]. Die Hierarchien und die Gruppenbildung spielen auch bei der Kontrolle der Emergenz eine große Rolle. So ist jeder Gruppe mindestens ein Kontroll-Knoten direkt übergeordnet. Der Kontroll-Knoten arbeitet dabei im Wesentlichen als Beobachter und greift nur zu Optimierungszwecken ein. Das Ziel ist dabei eine globale Optimierung der Ressourcennutzung. Bei Eingriffen gilt, dass die Entscheidung des Kontroll-Knoten eine höhere Priorität hat als die lokalen Entscheidungen der Einzelkomponenten eines Teilsystems. Somit sind die Ziele der höheren Ebene als wichtiger eingestuft. Auf ihrer eigenen Ebene agieren die Kontroll-Knoten wie reguläre IoT-Devices. Wie auch die IoT-Devices dürfen sie mit den Devices der gleichen Hierarchieebene ihrer Gruppe direkt kommunizieren. Dadurch entstehen netzwerkseitig vollständige Subgraphen. Dies ist vergleichbar mit einer *Peer-to-Peer* Architektur [Tan03, S. 7]. Die direkte Kommunikation hat eine lokale Stromversorgung zum Ziel. Mit lokaler Stromversorgung ist gemeint, dass es wünschenswert ist, in der Gruppe produzierten Strom auch direkt zu verbrauchen. Im Fall einer

zentralen Energieversorgung kann es in dem Szenario aus Abbildung 2.3 auf Seite 9 beispielsweise auftreten, dass der erhöhte Stromverbrauch des mittleren Hauses direkt ein Defizit im öffentlichen Netz verursacht, wodurch die Leistung eines Kraftwerks hochgefahren wird. Als weitere Folge wird die Überproduktion der PVA in das öffentliche Netz eingespeist und verursacht eine Überproduktion im gesamten Netz. Die lokale Verhandlung bietet somit einen Vorteil. Diese lokale Arbeitsweise spiegelt sich auch in der Selbstorganisation der Teil-Systeme wieder.

3.1.2. Unterschiede

Ein für den Simulator eher weniger relevanter Unterschied ist, dass das Management-System erst ab der Größe eines Micro-Grids eingeführt wird. Damit besteht keine direkte Verbindung zwischen dem Management-System und den Komponenten eines Nano-Grids. Das Management-System ist in den Szenarien der DevOps-Schicht zugeordnet und liegt damit logisch über dem Gesamtsystem. Das Bereitstellen eines Management-Systems für jedes einzelne Haus ist schlicht zu aufwendig. Ein weiterer Unterschied ist in den verwendeten smarten Komponenten zu finden. So werden auf der Nano-Grid Ebene IoT-Devices und Kontroll-Knoten genutzt. Auf der Micro- und Urban-Grid Ebene findet man nur Kontroll-Knoten vor. Wie sich die verschiedenen IoT-Devices und die Kontroll-Knoten voneinander unterscheiden, beschreibt der Abschnitt 3.3 auf Seite 34. Weiterhin zu beachten sind die Abhängigkeiten bezüglich des Aufbaus. Höhere Ebenen basieren auf den darunter liegenden Ebenen und erhalten ihre Daten von diesen. Ein Urban-Grid kann nicht funktionieren, sofern keine IoT-Devices vorhanden sind, umgekehrt jedoch schon. Untere Ebenen dürfen somit nicht in ihrer grundlegenden Funktion gestört werden, wenn eine höhere Ebene ausfällt. Der Ausfall führt lediglich dazu, dass die Emergenz unkontrolliert bleibt und einige Optimierungen nicht stattfinden.

3.1.3. Analyseergebnis

Der unmittelbar vorangegangene Vergleich der drei Szenarien hat detailliert die Gemeinsamkeiten und Unterschiede dargestellt. Wie die Analyse aufzeigt, besteht zwischen dem Aufbau und den Komponenten eine Differenz, der in der Konzeptentwicklung Aufmerksamkeit geboten werden muss. So müssen bei der Erstellung eines Szenarios gewisse Abhängigkeiten eingehalten und auf den korrekten Einsatz der Devices geachtet werden. Die Analyse hat jedoch auch gezeigt, dass die Grids viele Gemeinsamkeiten aufweisen. Diese spiegeln sich unter anderem in der lokalen Arbeitsweise und der Überwachung einer Schicht durch die jeweils übergeordnete Ebene wieder. Eine entscheidende Gemeinsamkeit stellt die Abbildung der smarten Komponenten auf (software) Agenten dar. Die Agenten bilden eine modulare Einheit

und können flexibel in den Szenarien eingesetzt werden, miteinander kommunizieren und sich gegenseitig überwachen. Zwar machen viele interagierende Agenten das System komplexer, da diese nach Definition eine gewisse Autonomie haben und eigenständig Verhalten entwickeln können. Dieses neue, emergente Verhalten kann sich allerdings auch als nützlich erweisen (siehe Unterabschnitt 2.2.2 auf Seite 14). Mit der ABMS ist zudem eine Methode vorhanden, um ein geeignetes Konzept für einen Simulator zu entwickeln, mit dem das emergente Verhalten zum Teil abgeschätzt und reproduziert werden kann. Die ABMS wird im Verlauf dieses Kapitels Schritt für Schritt angewendet. Bevor die eigentliche Konzeptentwicklung beginnt, werden in dem nächsten Abschnitt die genauen Anforderungen an den Simulator spezifiziert.

3.2. Anforderungen an den Simulator

In diesem Abschnitt der Arbeit werden die Anforderungen an den softwarebasierten Simulator definiert. Die Anforderungen sind abgeleitet aus den in Abschnitt 2.1 auf Seite 5 vorgestellten Szenarien, den Grundlagen über selbstorganisierende, emergente Systeme aus Abschnitt 2.2 auf Seite 12 sowie der unmittelbar vorangegangenen Szenarien-Analyse.

Die Definitionen der Anforderungen folgen der in Abbildung 3.1 auf der nächsten Seite dargestellten Schablone für textuelle Anforderungen. Eine Anforderung ist nach der Schablone in fünf Teile untergliedert. Zuerst erhält die Anforderung eine eindeutige fortlaufende Nummer und einen Namen. In dem zweiten Schritt kategorisiert das Schema die Anforderungen in MUSS und SOLL. Die MUSS-Anforderungen sind dabei Bedingungen an die Agenten des Simulators. Bleibt eine dieser Anforderungen unerfüllt, kann dies dazu führen, dass der Simulator später nicht das gewünschte Agentenverhalten widerspiegelt. Die SOLL-Anforderungen beziehen sich auf weiterführende, wünschenswerte Funktionalitäten und die Qualität der entwickelten Software. Unerfüllte Anforderungen dieser Kategorie schränken das Verhalten der interagierenden Komponenten nicht ein, sondern mindern lediglich den Funktionsumfang und die Anpassbarkeit des späteren Simulators. Der Kategorisierung folgt die Formulierung der Anforderung. In dem vierten Teil wir die Anforderung genauer erläutert. Zuletzt wird eine Lösungsidee skizziert.

Nach dem beschriebenen Schema wurden insgesamt 14 Anforderungen an den Simulator definiert. Die Anforderungen gliedern sich dabei in zwei Kategorien: (1) Funktionale Anforderungen und (2) Qualitative Anforderungen. Mit den in diesem Abschnitt gestellten Anforderungen sowie der vorangegangenen Szeanrien-Analyse kann in den anschließenden drei Abschnitten ein theoretisches Konzept für einen softwarebasierten Simulator entwickelt werden.

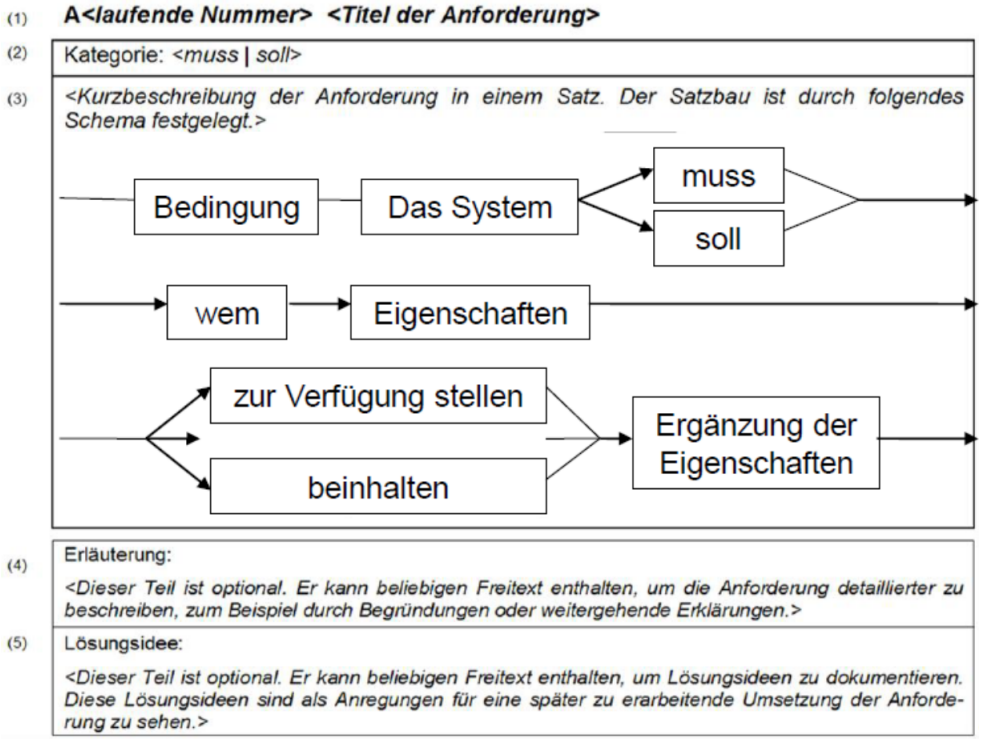


Abbildung 3.1.: Schablone für textuelle Anforderungen. Die Grafik wurde von der AG SSE der Universität Hildesheim zur Verfügung gestellt und ist angelehnt an [Rup16]. Die Schablone zeigt den schematischen Aufbau der in diesem Abschnitt definierten Anforderungen an den Simulator. Eine textuelle Anforderung gliedert sich dabei in die fünf Abschnitte Nummer und Titel, Kategorie, Kurzbeschreibung, Erläuterung (optional) und Lösungsidee (optional).

3.2.1. Funktionale Anforderungen

Die Anforderungen (F1) bis (F10) stellen die funktionalen Anforderungen dar. Die Anforderungen (F1) bis (F5) definieren die Abbildung der realen IoT-Devices auf Software sowie das Vorhandensein von Objekten, die das Verhalten der IoT-Devices widerspiegeln und miteinander interagieren. In den drei darauffolgenden Anforderungen werden Bedingungen an den Simulationsablauf und die Bereitstellung und Visualisierung der Simulationsdaten gestellt. Zuletzt beinhalten die Anforderungen (F9) und (F10) die Interaktion zwischen Anwender und Simulator.

(F1) Abbildung der IoT-Devices eines Electric-Grid

Kategorie: Muss
Der Simulator muss die realen IoT-Devices eines Electric-Grid Szenarios auf Software abbilden.
Erläuterung: Die unmittelbar vorangegangene Szenarien-Analyse hat innerhalb der Grids smarte Komponenten identifiziert. Eine detaillierte Spezifikation und Kategorisierung erfolgt in Abschnitt 3.3 auf Seite 34.
Lösungsidee: Wie in der vorangegangen Analyse beschrieben, können die IoT-Decvices als Agenten aufgefasst werden. Die Lösung besteht nun darin, jedes Device als einen Agenten abzubilden.

(F2) Abbildung des Verhaltens

Kategorie: Muss
Die Agenten müssen die Aufgaben und das Verhalten von IoT-Devices eines Electric-Grid Szenarios widerspiegeln.
Erläuterung: Das zu simulierende Verhalten ist detailliert in Abschnitt 3.3 auf Seite 34 dargestellt. Insbesondere muss die in der Szenarien-Analyse erwähnte Kommunikation abgebildet werden.
Lösungsidee: In (F1) wurden als Lösungsansatz Agenten vorgestellt. Ein Agent besitzt interne Verhaltensregeln (vgl. Abschnitt 2.3 auf Seite 16). Durch geeignete Spezifikationen dieser Regeln ist der Agent in der Lage, das Verhalten von einem IoT-Device eines Electric-Grids abzubilden.

(F3) Interaktion der Agenten

Kategorie: Muss
Die Agenten müssen nach der Aktivierung miteinander interagieren, sofern Kommunikationspartner vorhanden sind.

Erläuterung: Die Aufgabe der Agenten ist die Verhandlung der Stromflüsse. Dabei soll Strom von den Erzeugern zu den Konsumenten fließen. Die Bedingungen eines Stromflusses werden vorab unter den Agenten mittels Kommunikation geklärt. Die Interaktionen finden nur statt, wenn Kommunikationspartner existieren und die Agenten ihre jeweiligen Partner kennen.
Lösungsidee: Zu Beginn der Simulation sowie bei Änderungen des Agentenbestandes während der Simulation werden den Agenten ihre Kommunikationspartner mitgeteilt, sofern Partner existieren. Wird ein Agent gestartet und befindet sich im aktiven Zustand, besteht sein Verhalten weitgehend aus dem Senden und Empfangen von Nachrichten. Über diesen Nachrichtenaustausch verhandeln die Agenten miteinander und legen die Stromflüsse fest.

(F4) Autonome Agenten

Kategorie: Muss
Die Agenten müssen autonom handeln.
Erläuterung: Nach Simulationsstart interagieren die in (F1) beschriebenen Agenten miteinander, um Stromflüsse zu verhandeln (vgl. (F2)). Dabei handeln die Agenten frei und unabhängig von den anderen Agenten gemäß des implementierten Verhaltens. Das Handeln eines Agenten soll auch nicht von Agentenausfällen beeinflusst werden, insbesondere nicht durch den Ausfall des zugeordneten Kontroll-Knoten. Ausnahmen davon bilden, wenn der Agent selbst oder ein aktueller Interaktionspartner ausfällt. Durch die Interaktionen kann emergentes Verhalten entstehen.
Lösungsidee: Zum Erreichen der Autonomie läuft jeder Agent in einem separaten Thread. Ein Thread ist ein leichtgewichtiger Ausführungsstrang eines Programms (vgl. [WA05], S- 234). Dadurch ist Unabhängigkeit zwischen den Agenten gegeben. In die Kommunikation und das emergente Verhalten wird vorerst nicht eingegriffen.

(F5) Begrenztes Handeln

Kategorie: Muss
Die Agenten müssen die vorgegebenen Grenzen einhalten.

Erläuterung: Folgende Grenzen sind den Agenten gesetzt: • Eingrenzung der Kommunikationspartner gemäß der in Abschnitt 2.1 auf Seite 5 vorgestellten Hierarchie. • Vorgabe des Kommunikationsprotokolls Diese Anforderung stellt eine Einschränkung der in (F3) geforderten Autonomie dar. Die Einschränkung ist insofern berechtigt, da Agenten in einem realen Electric-Grid diesen ebenfalls unterlegen sind.
Lösungsidee: Durch die hierarchische Aufteilung der Simulationsumgebung sind die möglichen Kommunikationspartner determiniert. Das übergeordnete Device kennt untergeordneten Instanzen und propagiert dieses Wissen bei Simulationsbeginn. So ist sichergestellt, dass ein Gerät nur die relevanten Geräte kennt. Der Ablauf des vorgeschriebenen Kommunikationsprotokolls wird in den Agenten implementiert.

(F6) Simulationsablauf

Kategorie: Soll
Die Simulation soll ohne Unterbrechungen laufen.
Erläuterung: Mit dieser Anforderung ist gemeint, dass kein vorab gesetztes Ende der Simulation existiert. Die Simulation kann jederzeit durch den Anwender beendet werden.
Lösungsidee: Zu Beginn wird für jeden Agenten ein separater Thread gestartet. In einer Endlosschleife senden die Agenten nun Nachrichten oder fragen eingegangene Daten ab und reagieren auf diese. Der Anwender kann die Simulation über das *Graphical User Interface* (GUI) oder durch Schließen des Programms beenden.

(F7) Darstellung der Simulationsdaten

Kategorie: Soll
In der GUI sollen die relevantesten Simulationsdaten dargestellt werden.

Erläuterung: Zu den wichtigen Simulationsdaten gehören die vorhanden Agenten, deren aktueller Zustand und die ausgetauschten Nachrichten. Der Zustand der Agenten ist dargestellt über den AN-/ AUS-Status und individuelle Attribute wie Verbrauch, Erzeugung oder Ladestand. Somit wird eine ausgehende Schnittstelle gefordert, die vorerst über die GUI abgebildet werden soll. Diese Schnittstelle ermöglicht im späteren Verlauf des Projekts der Kontroll-Schicht die Überwachung der Agenten und deren Verhalten.
Lösungsidee: In der bereitgestellten GUI sind während der Simulation alle Agenten aufgeführt. Über einen Klick auf einen Agenten kann der Zustand abgefragt werden. Für die Darstellung der Nachrichten beinhalten die GUI einen separaten Anzeigebereich, welcher kontinuierlich aktualisiert wird.

(F8) Bereitstellung eines Debug-Modus

Kategorie: Soll
Das System soll dem Anwender einen Debug-Modus bereitstellen.
Erläuterung: Der Debug-Modus soll dem Anwender beispielsweise bei der Überprüfung des Simulationsablaufs behilflich sein. Diese Anforderung ist eine Erweiterung von (F6). Analog zu (F6) wird implizit eine Schnittstelle aus dem Simulator heraus gefordert.
Lösungsidee: Beim Start des Programms kann der Anwender zwischen dem normalen Modus und dem Debug-Modus entscheiden. Im Debug-Modus werden alle ausgetauschten Nachrichten in ein separates Log-File geschrieben. Zudem beinhalten die in der GUI dargestellten Nachrichten erweiterte Informationen.

(F9) Ändern der Simulationsparameter

Kategorie: Soll
Der Simulator soll dem Anwender die Möglichkeit der Einflussnahme zur Verfügung stellen. Die Einflussnahme soll während des laufenden Betriebs stattfinden.
Erläuterung: Der Anwender soll die Möglichkeit haben, die IoT-Devices an- und abzuschalten. Weiterhin müssen Parameter wie der Verbrauch, Erzeugung und Ladestand änderbar sein. Diese Anforderung erzwingt eine Schnittstelle in das System. Über diese Schnittstelle kann die Kontroll-Schicht später Änderungen des Agentenverhaltens anweisen.

Lösungsidee: Über die GUI kann der Anwender das jeweilige IoT-Devices anklicken. Nach der Auswahl stehen dem Anwender die gewünschten Parameter zur Verfügung und können geändert werden. Während des Änderungsprozesses wird die Anwendung nicht gestoppt.

(F10) Hinzufügen und Entfernen von Devices

Kategorie: Soll
Der Anwender soll die Möglichkeit haben, IoT-Devices hinzuzufügen oder zu entfernen.
Erläuterung: Mit dieser Funktion kann die Integration von neuen IoT-Devices in ein Szenario simuliert werden. Das Hinzufügen und Entfernen bedarf gegebenenfalls einer Simulationsunterbrechung, was eine Einschränkung der Anforderung (F3) zur Folge hat.
Lösungsidee: Die Devices werden während der Simulation in einer Liste verwaltet. Für das Hinzufügen erstellt der Anwender eine neue Instanz, welche der Liste hinzugefügt wird. Analog können Instanzen aus der Liste entfernt werden.

3.2.2. Qualitative Anforderungen

In (Q1) bis (Q4) sind die qualitativen Anforderungen dargestellt. In der ersten Anforderung dieser Kategorie wird das Reaktionsverhalten des laufenden Simulators, insbesondere während und nach Interaktionen mit dem Anwender limitiert. Zudem wird in dieser Kategorie gefordert, dass das Simulator-Konzept wart- und erweiterbar ist und die theoretischen Grundzüge auf verwandte Domänen anwendbar sind.

(Q1) Reaktionsverhalten des Simulators

Kategorie: Soll
Das System soll ohne eine nennenswerte Unterbrechung der Simulation auf Änderungen der Simulationsparameter reagieren. Eine Ausnahme bildet hierbei das Hinzufügen und Entfernen von IoT-Devices.

Erläuterung: Um die Anforderung (F3) nicht zu verletzen, sollen Änderungen, wie das Aus-/Anschalten der IoT-Geräte oder eine Änderung der Device-Parameter keine Unterbrechung der Simulation zur Folge haben. Die Änderungen sollen unmittelbar in die Simulation einfließen. Beim Hinzufügen und Entfernen von IoT-Devices wird die Simulation kurzzeitig pausiert.
Lösungsidee: Bei Änderungen des Gerätezustandes wird der geänderte Parameter für einen Bruchteil gesperrt. Die Änderung betrifft jedoch nur das eine IoT-Device und muss nicht weiter propagiert werden. Beim Hinzufügen/Entfernen betrifft die Änderung das neue/gelöschte IoT-Device, das Simulationsobjekt, das übergeordnete IoT-Device sowie alle Kommunikationspartner. Die Änderung muss somit an viele Geräte propagiert werden. Zudem muss ein Thread erstellt oder gelöscht werden. Damit in dieser Phase keine Fehler passieren, wird der reguläre Simulationsablauf pausiert.

(Q2) Wartbarkeit

Kategorie: Soll
Das erstellte Simulator-Konzept soll wartbar sein.
Erläuterung: Eine hohe Wartbarkeit ermöglicht eine rasche Fehlerbehebung und Anpassung an veränderte Umgebungen. Mit Wartbarkeit ist in diesem Zusammenhang beispielsweise die Anpassung von Kommunikationsprotokollen oder Schnittstellen sowie die Behebung von Fehlern gemeint.
Lösungsidee: Zur Erreichung der Wartbarkeit werden Modularisierung, Kapselung und eine lose Kopplung verwendet. Weiterhin wird der Programmcode ausführlich dokumentiert. Für einen Integrationstest im Anschluss an die Wartungsarbeiten, werden automatisierte Tests bereitgestellt.

(Q3) Erweiterbarkeit

Kategorie: Soll
Das erstellte Simulator-Konzept soll erweiterbar sein.
Erläuterung: Mögliche Erweiterungen sind neue Device-Arten und Verhaltensänderungen der vorhandenen Devices.

Lösungsidee: Zur Erreichung der Erweiterbarkeit werden Modularisierung, Kapselung und eine lose Kopplung verwendet. Weiterhin wird der Programmcode ausführlich dokumentiert. Weitere Device-Arten können in die Vererbungshierarchie integriert werden.

(Q4) Anwendbarkeit auf verwandte Domänen

Kategorie: Soll
Das erstellte Simulator-Konzept soll auf andere Szenarien und Domänen anwendbar sein.
Erläuterung: Das Verhalten der implementierten Agenten ist an ein Electric-Grid Szenario gebunden. Das genaue Szenario wird dabei nicht spezifiziert, indem die IoT-Devices grob in die Klassen Verbraucher, Erzeuger, Speicher und Hausanschluss eingeordnet werden. Eine Übertragbarkeit des Konzepts soll bei der Herangehensweise sowie der Hierarchiebildung zum Tragen kommen. Dabei sollen die theoretischen Grundlagen des Konzepts nach Anpassung der Regeln und der Agenten auf verwandte IoT-Systeme anwendbar sein. Andere Domänen sind beispielsweise die smarte Wasserversorgung.
Lösungsidee: Die hierarchische Trennung zwischen Kontroll- und IoT-Schicht erlaubt eine Austauschbarkeit des IoT-Szenarios. So können auf der IoT-Ebene neue Devices hinzukommen, ohne die Kontroll-Schicht direkt zu betreffen und umgekehrt. Ebenso erleichtert die Trennung die Anpassbarkeit der einzelnen Schichten. Die Theorie lässt sich grob zusammenfassen in agentenbasierte Modellierung, um die Emergenz und Selbstorganisation darzustellen, die Hierarchiebildung für die Kommunikation und die Verwendung einer übergeordneten Kontroll-Schicht. Die ABMS lässt sich dabei auf jede Domäne anwenden, in der die Komponenten auf Agenten abgebildet werden können. Ist dies der Fall, kann eine Hierarchie innerhalb der Agenten erstellt werden. Innerhalb dieser Hierarchie können dann bestimmte Ebenen als Kontroll-Schichten fungieren. So ist es möglich, dass sowohl beispielsweise eine intelligente Steckdose aus einem Electric-Grid abgebildet werden kann, als auch ein intelligenter Wasserhahn oder ein Drucksensor.

3.3. Enwicklung eines Agentenbasierten Modells

In diesem Abschnitt wird die ABMS auf ein Electric-Grid angewendet, um ein agentenbasiertes Modell des Systems zu entwickeln. Dazu werden (1) die Menge der Agenten mit Attributen und Verhaltensweisen, (2) die Agentenbeziehungen und Interaktionen und (3) die Umwelt der Agenten genau spezifiziert. Die Spezifikation

orientiert sich dabei an den in Unterabschnitt 2.3.2 auf Seite 19 vorgestellten sieben Kernfragen der ABMS. Wichtig für die Verwendung der ABMS ist, dass innerhalb des Systems Agenten identifizierbar sind. Dieses Kriterium ist nach Unterabschnitt 3.1.1 auf Seite 24 erfüllt, da die smarten Komponenten eines Electric-Grids auf Agenten abgebildet werden können.

3.3.1. Problemstellung

Mit der ersten Frage wird geklärt, welches konkrete Problem innerhalb des Systems besteht und mittels des zu entwickelnden agentenbasierten Modells gelöst werden soll. Dabei ist zudem von Interesse, welchen Mehrwert die agentenbasierte Modellierung gegenüber existierenden Ansätzen bietet. Mit dem Simulator-Konzept sollen die Abläufe innerhalb der IoT-Schicht eines Electric-Grid dargestellt werden. Dazu werden insbesondere die Verhaltensweisen von IoT-Devices, welche etwa in Gebäuden oder Fabriken eingesetzt werden, betrachtet. Des Weiteren werden die Komponenten der Kontroll-Schicht berücksichtigt, damit Interaktionen zwischen Haushalten und auf höherer Ebene zwischen Stadtteilen möglich sind. Mit dem resultierenden Simulator sollen mögliche emergente Verhaltensweisen sichtbar werden. Weiterhin können im späteren Verlauf Regeln für die Kontroll-Schicht und das Einspielen von Änderungen getestet werden. Die zentrale Frage dabei ist, mit welchen Einschränkungen das IoT-Ökosystem softwarebasiert simulierbar ist. Die agentenbasierte Simulation liefert einen Mehrwert, indem die Systemparameter während des Betriebs änderbar sind. Durch diese Änderungen treten in der Regel Reaktionen innerhalb des Systems auf. Durch geeigente Simulationsverfahren können diese Reaktionen für den Anwender visualisiert und somit beobachtbar gemacht werden. Weiterhin besteht die Chance, mittels Simulation mögliche Emergenzen aufzudecken (siehe Unterabschnitt 2.2.2 auf Seite 14). Wie in Unterabschnitt 2.3.3 auf Seite 21 erläutert, trägt agentenbasierte Simulation zu einem tieferen Verständnis komplexer Systeme bei.

3.3.2. Agenten des Modells

Die zweite Frage unterstützt bei der Identifikation der benötigten Agenten und deren statischen und dynamischen Attributen. Für die Spezifikation der Agenten müssen die smarten Komponenten eines Electric-Grid identifiziert werden. Dazu wird erneut Bezug auf die Szenarien aus Abschnitt 2.1 auf Seite 5 sowie die unmittelbar vorangegangene Analyse genommen. Im Wesentlichen lassen sich die smarten Komponenten in zwei Kategorien unterscheiden. Diese Kategorien sind die reinen IoT-Devices und die Kontroll-Knoten. Bei den Kontroll-Knoten ergibt sich eine feinere Unterteilung in die Kategorien Energiemanager, SNN und die oberste Kontroll-Schicht. Die identifizierten IoT-Devices lassen sich näher kategorisieren in Verbraucher, Erzeuger,

Energiespeicher und Hausanschluss. Beispiele für diese Kategorien sind ein Smart Plug als Verbraucher, eine PVA und ein Windpark als Erzeuger, eine Batterie als Energiespeicher und eine smarte Steckdose als Hausanschluss. Tabelle 3.1 und Tabelle 3.2 auf der nächsten Seite stellen die Agenten mit ihren zugehörigen statischen und dynamischen Attributen übersichtlich dar.

Agent	**Attributtyp**	**Attribut**
Verbraucher	statisch	• ID • Verbrauch
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordneter Energiemanager
Erzeuger	statisch	• ID • Maximale Erzeugung • Erneuerbarer Erzeuger
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordneter Energiemanager • Momentane Erzeugung • Verfügbare Strommenge
Energiespeicher	statisch	• ID • Kapazität
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordneter Energiemanager • Ladestatus • Verfügbare Strommenge
Hausanschluss	statisch	• ID
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordneter Energiemanager • Aktueller Strompreis

Tabelle 3.1.: IoT-Devices und die statischen und dynamischen Attribute. Die Tabelle illustriert die statischen und dynamischen Attribute der IoT-Devices eines Electric-Grids. Für die IoT-Devices ergibt sich eine Unterteilung in Verbraucher, Erzeuger, Energiespeicher und Hausanschluss. Beispiele für diese Kategorien sind ein Fernseher, eine PVA, eine Batterie oder eine Steckdose.

Agent	Attributtyp	Attribut
Energiemanager	statisch	• ID
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordneter SNN • Untergeordnete IoT-Devices • Verbindungen der IoT-Devices • Energieverbrauch des Gebäudes • Energieerzeugung des Gebäudes • Regeln für die Kontrollaufgabe
SNN	statisch	• ID
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Übergeordnete Kontroll-Schicht • Untergeordnete Energiemanager • Verbindungen der Energiemanager • Energieverbrauch des Stadtteils • Energieerzeugung des Stadtteils • Regeln für die Kontrollaufgabe
Oberste Kontroll-Schicht	statisch	• ID
	dynamisch	• Aktivitätsstatus • Kommunikationspartner • Aktive Verbindungen • Untergeordnete SNN • Verbindungen der SNN • Energieverbrauch der Stadt • Energieerzeugung der Stadt • Regeln für die Kontrollaufgabe

Tabelle 3.2.: Kontroll-Knoten und die statischen und dynamischen Attribute. In der Tabelle sind die Arten von Kontroll-Knoten abgebildet. Ein Kontroll-Knoten kann kategorisiert werden in einen Energiemanager, einen SNN oder die Oberste Kontroll-Schicht, von der nur eine Instanz existiert. Die Energiemanager sind dabei den IoT-Devices, die SNN den Energiemanagern und die Oberste Kontroll-Schicht den SNN übergeordnet.

Ein Agent ist nach Definition eine eindeutig identifizierbare Instanz (siehe Definition 2.3 auf Seite 17). Die Eindeutigkeit eines Agenten ist über die statische und nicht veränderbare ID in den beiden Tabellen abgebildet. Ein Agent verfügt weiterhin über einen änderbaren Aktivitätsstatus, welcher anzeigt, ob der Agent zum

jeweiligen Abfragezeitpunkt aktiv oder inaktiv ist. Zudem enthält jede Instanz anpassbare Informationen über seine umliegenden Kommunikationspartner, die aktiven Verbindungen und den übergeordneten Kontroll-Knoten. Die Kontroll-Knoten halten außerdem Informationen über die ihnen untergeordneten Agenten und deren Verbindungen sowie den momentanen Energieverbrauch und die Energieerzeugung der kontrollierten Gruppe. Für den späteren Einsatz wurde bereits das Hinzufügen eines Regelsets für die Kontrollaufgaben bedacht. Bezüglich ihrer Attribute weisen die Agenten somit einige Gemeinsamkeiten auf, welche in der späteren Architektur des Simulators berücksichtigt werden, indem diese Attribute in übergeordneten Klassen zusammengefasst werden. Die IoT-Devices enthalten außerdem individuelle Attribute, wie etwa die Kapazität eines Energiespeichers oder die momentane Erzeugung und verfügbare Strommenge eines Erzeugers. Die momentane Erzeugung ist ein Attribut, welches durch den Anwender angepasst werden kann. Die verfügbare Strommenge berechnet sich aus der Differenz der momentanen Erzeugung und der bereits an andere Agenten abgegebenen Strommenge.

3.3.3. Umwelt und Interaktionen mit der Umwelt

Im Anschluss wird mit der Beantwortung der dritten Frage die Umwelt der Agenten sowie die Interaktionen der Agenten mit ihrer Umwelt spezifiziert. Dabei wird zudem beschrieben, inwiefern sich die Agenten innerhalb ihrer Umwelt bewegen können. Die Agentenumwelt ist in diesem Fall das IoT-Ökosystem eines Electric-Grids, also die Umgebung und die Gebäude eines smarten Stromnetzes. Die Agenten interagieren mit dem Electric-Grid in der Weise, dass sie durch ihre gemeinsamen Aktionen versuchen, das Stromnetz stabil zu halten. Diese Stabilität wird durch die Einspeisung von Strom bei Unterbelastung oder die Abnahme von Strom bei Überproduktion erreicht. Damit geschieht diese Interaktion der Agenten mit ihrer Umwelt eher indirekt. Zudem spielt die äußere Einflussnahme, wie etwa durch die Sonneneinstrahlung oder den Wind bei erneuerbaren Erzeugern wie etwa einer PVA, eine große Rolle. Aufgrund des Schwerpunkts der Arbeit werden die Einflüsse der Natur abstrahiert und über Parameteränderungen durch den Anwender abgebildet. Damit stellt der Anwender einen Teil der Umwelt dar. Der Anwender interagiert in den folgenden Fällen mit dem Agenten: (1) Agent an-/abschalten, (2) Verbrauch/ Erzeugung ändern und (3) Agent hinzufügen oder entfernen. Weiterhin haben die Gebäude einen indirekten Einfluss, indem sie die Agenten in Gruppen isolieren und die jeweiligen Kommunikationspartner einschränken. Eine weitere Interaktion mit der Umwelt stellt das Ein- und Ausschalten von Verbrauchern oder das Hinzufügen und Entfernen von Agenten innerhalb des Systems durch den Anwender dar. Die smarten Komponenten sind an ihren Platz bzw. ihre Gruppe gebunden und können sich nicht innerhalb der Umwelt bewegen. Lokalisation spielt somit vorerst keine

Rolle. In der Diskussion wird in Abschnitt 6.3 auf Seite 73 jedoch mit dem mobilen Verbraucher ein Agent vorgestellt, welcher seine Position in der Umwelt ändern kann. Ein mobiler Verbraucher kann etwa ein Elektrofahrzeug oder ein Smartphone sein. Ein Agent dieses Typs soll in der Lage sein, sich innerhalb des Systems zu bewegen (eigenständig oder durch Fremdeinwirkung) und dabei den jeweils übergeordneten Energiemanager zu wechseln. In Erweiterungen kann zudem der aktuelle Strompreis einen Einfluss auf die Agenten haben, indem z.B. bei hohen Strompreisen bevorzugt in das öffentliche Stromnetz eingespeist wird.

3.3.4. Agentenverhalten

Mit der vierten Frage wird auf das individuelle Agentenverhalten Bezug genommen. Dabei soll spezifiziert werden, welche Entscheidungen die Agenten treffen, nach welchen Verhaltensweisen sie handeln und welche Aktionen die Agenten unternehmen. Eine grundlegende Verhaltensweise ist die Kommunikation mit anderen benachbarten Agenten. Der Zweck dieser Interaktion ist die Verhandlung der Stromflüsse innerhalb der Nachbarschaft. Mit der Nachbarschaft sind die Agenten der gleichen Gruppe, beispielsweise eines Nano-Grids, gemeint. Die Kommunikation setzt dabei voraus, dass die Agenten ihre umliegende Nachbarschaft kennen. Die Distribution der Gruppenzugehörigkeiten kann beispielsweise Aufgabe der direkt übergeordneten Kontroll-Schicht sein oder über Subnetzzugehörigkeiten geschehen. Weiterhin teilen alle Agenten ihrem zugeordneten Kontroll-Knoten die jeweilige Kommunikationsabsicht (*brauche Strom* oder *habe Strom*) mit und informieren diesen über eine erfolgreiche Verhandlung und das Ende des Stromflusses. Diese Verhaltensweisen betreffen sowohl die IoT-Schicht, in der Versorger und Abnehmer direkt Vereinbarungen treffen, als auch die Kontroll-Schichten, auf denen Über- und Unterproduktionen eines Gebäudes oder Straßenzugs eine Rolle spielen. Kommunikationspartner sind dabei die Agenten der selben Gruppe, also im Fall eines Nano-Grids alle smarten Komponenten des Gebäudes, im Fall eines Micro-Grids die Energiemanager des Straßenzugs und bei einem Urban-Grid die SNNs der smarten Stadt. Ein weiteres gemeinsames Verhalten ist, dass sich ein neuer Agent bei seiner zuständigen Kontroll-Schicht registriert, sobald er in das Szenario eingebunden wird. Dieses Verhalten ist insbesondere wichtig bei möglichen mobilen Agenten, welche regelmäßig ihre Gruppe wechseln. Im ausgeschalteten Zustand haben die Agenten kein Verhalten. Die Kommunikation verläuft über ein einheitliches Protokoll, welches in dem nachfolgenden Abschnitt ausführlich vorgestellt wird. Prinzipiell soll es möglich sein, dass die Agenten kommunizieren und interagieren können, obwohl die Kontroll-Schicht ausgefallen ist. Für die weitere Beantwortung dieser Frage werden die Agenten separat betrachtet. Die nachfolgende Auflistung zeigt das individuelle Agentenverhalten der in der zweiten Antwort erwähnten Kategorien:

- **Verbraucher** Ein Agent aus der Kategorie Verbraucher fragt, sobald er eingeschaltet ist bzw. durch den Anwender eingeschaltet wird, aktiv bei den umliegenden IoT-Devices nach Strom. Zudem reagiert der Verbraucher auf Anfragen aus seinem Umfeld, sofern diese von Stromlieferanten kommen und der Verbraucher aktuell nicht versorgt wird. Ein Verbraucher kann genau einen Versorger gleichzeitig haben.
- **Erzeuger** Ein Erzeuger beginnt aktiv die Kommunikation, wenn er weitere Stromabnehmer versorgen kann, indem er eine Nachricht mit der verfügbaren Strommenge an seine Nachbarschaft sendet. Zudem reagiert der Erzeuger auf Nachrichten von Verbrauchern der Gruppe, wenn er Strom abgeben kann. Die Anzahl der Abnehmer ist durch die verfügbare Strommenge limitiert.
- **Energiespeicher** Ein Energiespeicher kann ebenfalls kommunizieren, startet jedoch keine Verhandlungen von selbst, sondern reagiert auf Nachrichten aus seinem Umfeld. Dabei kann der Energiespeicher einerseits auf Erzeuger reagieren, wenn der Ladestand unterhalb eines festgelegten Werts ist. Der Ladestand erhöht sich über die Dauer der Verbindung. Andererseits kann der Energiespeicher auf Verbraucher reagieren, wenn der Ladestand oberhalb eines festgelegten Werts liegt. Dabei verringert sich der Ladestand. Der Lade- und Entladevorgang des Speichers verläuft von der Kommunikationsseite analog zu einer Stromversorgung zwischen Verbraucher und Erzeuger.
- **Hausanschluss** Ähnlich wie ein Energiespeicher hat ein Hausanschluss kein aktives Verhalten. Der Hausanschluss erhält Nachrichten bezüglich Stromabnahme oder Stromversorgung und reagiert auf diese, indem ein Einspeisen in das öffentliche Netz bzw. die Versorgung angeboten wird. Ein Hausanschluss kann beliebig viele Komponenten versorgen.
- **Energiemanager** Ein Energiemanager überwacht die untergeordneten IoT-Devices des zugewiesenen Gebäudes und ermittelt den Stromverbrauch und die Stromerzeugung. Sobald der Energiemanager eine Über- oder Unterversorgung identifiziert, kontaktiert er die Energiemanager der umliegenden Gebäude. Im Fall einer Unterversorgung nimmt der Manager dabei die Rolle des Verbrauchers ein und handelt nach dessen Verhalten. Bei einer Überversorgung agiert er analog zu einem Erzeuger. Weiterhin gehören zu dem Verhalten noch nicht spezifizierte Regeln für die Kontroll-Aufgaben.
- **SNN** Ein SNN kontrolliert die Stromflüsse zwischen den Gebäuden eines Stadtteils und überwacht dabei die benötigte und erzeugte Strommenge. Im Fall einer Unterversorgung kontaktiert der SNN die umliegenden Stadtteile und nimmt die Rolle des Verbrauchers ein. Bei einer Überversorgung agiert er analog zu einem Erzeuger. Weiterhin gehören zu dem Verhalten noch nicht spezifizierte Regeln für die Kontroll-Aufgaben.

- **Oberste Kontroll-Schicht** Die oberste Kontroll-Schicht hat zum jetzigen Zeitpunkt nur die Kontroll- und Überwachungsaufgabe. Die Regeln für dieses Verhalten werden erst im Verlauf des Projekts *DevOpt* spezifiziert.

Entscheidend bei der Spezifikation des Agentenverhalten ist, dass die Agenten möglichst autonom handeln. Dazu gehört zum einen, dass Agenten auf unteren Ebenen funktionieren, selbst wenn die Kontroll-Schicht ausfällt. Zum anderen soll ein Agent unabhängig von anderen arbeiten. Das Verhalten der Agenten wurde bewusst einfach gehalten, damit Komponenten auf der untersten Ebene, dem IoT-Layer, nicht zu viel Einfluss haben. Auf dieser Ebene soll lediglich eine Versorgung bzw. eine Verhandlung von Stromflüssen stattfinden. Optimierungen erfolgen durch den Eingriff höherer Instanzen. Weiterhin reicht, wie in Unterabschnitt 2.2.2 auf Seite 14 beschrieben, bereits einfaches Verhalten aus, um ein komplexes Gesamtsystem zu erhalten. Eine Alternative wäre, das Grundverhalten der Agenten zu erweitern und so Teile der Kontroll-Schicht, insbesondere bezüglich der Optimierung, in die IoT-Schicht zu verschieben. Dies würde die Kontroll-Schicht jedoch teilweise überflüssig machen und eventuell zu lokalen Optima führen. Von besonderem Interesse ist das durch die Interaktionen entstehende emergente Verhalten innerhalb des Gesamtsystems. Dieses kann jedoch vorab nicht erfasst und aufgeführt werden.

3.3.5. Agenteninteraktionen und deren Ausmaß

Die fünfte Frage betrachtet die Interaktionen der Agenten untereinander und bezieht dabei die Umwelt mit ein. Von Interesse ist dabei auch das Ausmaß der Interaktionen. Wie bereits erwähnt, interagieren die Agenten mittels Kommunikation miteinander. Durch die gemeinsame Kommunikation werden Stromflüsse lokal verhandelt. Die Auswirkungen betreffen somit vorerst nur die direkte Umgebung. Diese Umgebung hängt von der Größe der Ebene ab, auf der die Änderungen geschehen und kann dadurch in der Größe variieren. Beispielsweise sind in einem Nano-Grid nur wenige IoT-Devices involviert, in einem Micro-Grid kann die Interaktion bereits die Komponenten mehrerer Nano-Grids sowie die jeweiligen Energiemanager betreffen. Die Interaktionen mit der Umwelt wurden bereits in Unterabschnitt 3.3.3 auf Seite 38 erläutert. Die Auswirkungen dieser Interaktionen sind ebenfalls lokal.

3.3.6. Quellen für die Spezifikation

Mit der sechsten Fragen wird geklärt, welchen Quellen das Agentenverhalten und die übrigen Spezifikationen entstammen. Die Verhaltensweisen der Agenten sowie die Konfiguration der Umwelt wurden aus den in Abschnitt 2.1 auf Seite 5 vorgestellten Szenarien sowie der zugehörigen Analyse aus Abschnitt 3.1 auf Seite 23 abgeleitet.

Zudem fließen die definierten Anforderungen, insbesondere (F3), (F4) und (F5), in die Spezifikation ein. In diesen Anforderungen wird definiert, dass die Agenten interagieren sollen, autonom sind und das Ausmaß der Interaktionen begrenzt ist.

3.3.7. Validierung des Modells

Die letzte, siebte Frage nimmt Bezug auf die Validierung des Modells sowie des simulierten Agentenverhaltens. Für die Validierung des Konzepts werden die in Abschnitt 3.2 auf Seite 26 definierten Anforderungen auf Erfüllbarkeit untersucht. Der implementierte Simulator wird mittels der in Abschnitt 2.1 auf Seite 5 vorgestellten Szenarien getestet. Die Validierung des Simulator-Konzepts und der Implementierung ist Teil von Kapitel 5 auf Seite 63.

Die wichtigste Verhaltensweise eines Agenten ist die Möglichkeit zur Kommunikation. Damit Kommunikation stattfinden kann, benötigen die Agenten eine gemeinsame Sprache. Der nachfolgende Abschnitt thematisiert diese gemeinsame Sprache und stellt das entwickelte Kommunikationsprotokoll vor.

3.4. Kommunikationsprotokoll

Dieser Abschnitt greift das relevanteste Agentenverhalten, die Kommunikation mit Agenten der gleichen Gruppe, auf und erklärt im Detail den Kommunikationsablauf. Der Ablauf reicht von der Registrierung beim übergeordneten Kontroll-Knoten über den Beginn der Stromfluss-Verhandlungen bis zu der abgeschlossenen Übertragung. Das für den Kommunikationsablauf entwickelte Protokoll gliedert sich in vier Phasen. Diese sind (1) die Registrierung bei dem übergeordneten Kontroll-Knoten, (2) der Verhandlungsbeginn, (3) der eigentliche Stromfluss und (4) das Übertragungsende. Die erste Phase wird einmalig bzw. bei einem Gruppenwechsel ausgeführt. Die Phasen (2) bis (4) finden bei jeder Verhandlung erneut statt. Die vier Phasen bestehen jeweils aus einem Nachrichtenaustausch zwischen Agenten.

Abbildung 3.2 auf der nächsten Seite zeigt den Aufbau einer Nachricht. In einer Nachricht kann jeweils das Attribut *Kommunikationsgrund* gesetzt werden, welches die jeweilige Absicht der Nachricht spezifiziert. Insgesamt gibt es zehn verschiedene Werte für dieses Attribut. Diese lauten wie folgt:

- **SEARCH:** Über SEARCH versucht sich ein neu hinzugefügter Agent bei seinem zugeordneten Kontroll-Knoten zu registrieren.
- **CONTROL:** Mit einem CONTROL reagiert der zuständige Kontroll-Knoten auf ein SEARCH.

Nachricht
• Sender • Empfänger • Kommunikationsgrund • Inhalt

Abbildung 3.2.: Attribute einer Nachricht. Eine Nachricht beinhaltet Informationen über den Sender, den Empfänger, den Kommunikationsgrund und den eigentlichen Inhalt. Als Sender und Empfänger werden die IDs der entsprechenden Agenten gesetzt. Den Kommunikationsgrund wählt der sendende Agent aus einem der zehn Werte: SEARCH, CONTROL, REQUEST, PROPOSE, ACCEPT_PROPOSAL, AGREE, INFORM, PROPAGATE, CONFIRM und DISCONFIRM. Über den Inhalt der Nachricht spezifiziert der Agent beispielsweise die benötigte Strommenge.

- **REQUEST:** Ein Agent macht mit einem REQUEST auf sich aufmerksam, dass er entweder Strom benötigt oder eigens produzierten Strom abgeben kann.
- **PROPOSE:** Ein Agent kann mit einem PROPOSE einem anfragenden Agenten ein Angebot machen. Erhält ein Verbraucher beispielsweise eine REQUEST-Nachricht von einem Erzeuger, welcher Strom abgeben kann, antwortet der Verbraucher mit der benötigten Strommenge.
- **ACCEPT_PROPOSAL:** Stimmt ein Agent dem Angebot zu, bestätigt er dies dem bietenden Agenten mit einem ACCEPT_PROPOSAL.
- **AGREE:** Mit einem AGREE wird die Verhandlung erfolgreich abgeschlossen.
- **INFORM:** Über ein INFORM benachrichtigen untergeordnete Agenten ihren Kontroll-Knoten über eine erfolgreiche Verhandlung oder das Ende des Stromflusses.
- **PROPAGATE:** Über ein PROPAGATE wird der virtuelle Stromfluss zwischen zwei Agenten abgebildet.
- **CONFIRM:** Als Reaktion auf das PROPAGATE sendet der an dem Stromfluss beteiligte Agent ein CONFIRM.
- **DISCONFIRM:** Ein DISCONFIRM beendet den virtuellen Stromfluss.

Die Bezeichnung der Attributwerte ist, bis auf die Begriffe SEARCH und CONTROL angelehnt an die *Agent Communication Language* (ACL) [ON98]. Entwickelt wurde die ACL durch die *Foundation for Intelligent Physical Agents* (FIPA). Die FIPA hat die Standardisierung und Interoperabilität heterogener, interagierender Agenten

zum Ziel[1]. Die Phasen des Protokolls werden nun anhand von Beispielen erläutert. Die Beispiele zeigen jeweils ein Nano-Grid, bestehend aus einem Verbraucher, einem Energiespeicher (Batterie), einem Erzeuger und einem Kontroll-Knoten (Energiemanager). Der Verbraucher stößt in den Beispielen die Kommunikation an.

3.4.1. Registrierung bei dem Kontroll-Knoten

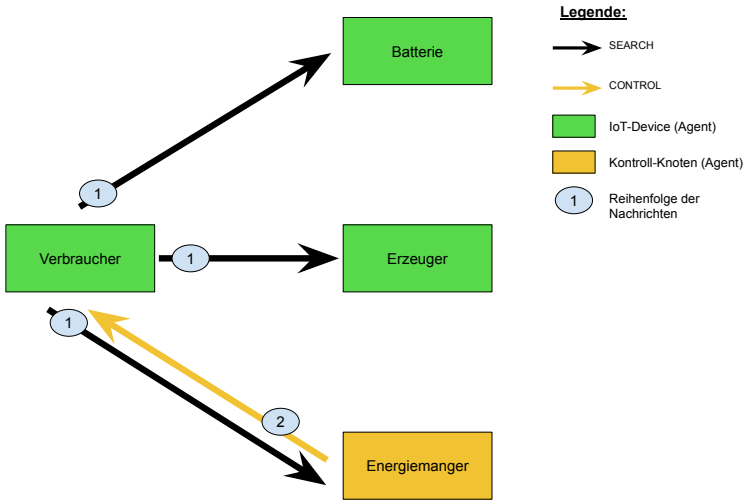


Abbildung 3.3.: Registrierung beim zuständigen Kontroll-Knoten. Wird ein neuer Agent in das Szenario hinzugefügt, registriert dieser sich selbstständig bei seinem zuständigen Kontroll-Knoten. Diese Abbildung visualisiert die Nachrichten, die bei dieser Registrierung versendet werden. Der neu hinzugefügte Agent ist der Verbraucher. Dieser sendet eine SEARCH-Nachricht an die Agenten seiner Nachbarschaft. Der zuständige Kontroll-Knoten, in diesem Fall ein Energiemanager, antwortet mit einem informierenden CONTROL. Nach diesem Nachrichtenaustausch kann Kontrolle ausgeübt werden.

Wenn ein Agent neu in das System hinzugefügt wird bzw. ein mobiler Agent seine Gruppe wechselt, ist es wichtig, dass insbesondere der neue übergeordnete Kontroll-Knoten diese Änderung erfährt. Ein nicht gesetzter Kontroll-Knoten bedeutet für das System, dass innerhalb dieses Teils nur bedingt Eingriffe und damit auch nur bedingt Optimierungen stattfinden können. Weiterhin kann der neue Agent keine Updates seitens des Management-Systems erhalten, da diese nach Abschnitt 2.1 auf Seite 5 über die zuständigen Kontroll-Knoten verteilt werden. Weiterhin ist es die Aufgabe

[1]Für weitere Informationen zur FIPA siehe: http://fipa.org/

des zuständigen Kontroll-Knotens, dem neuen Agenten die möglichen Kommunikationspartner mitzuteilen und die möglichen Kommunikationspartner über den neuen Agenten zu informieren. Damit können Interaktionen zwischen den Agenten nur stattfinden, wenn bei der Registrierung der zuständige Kontroll-Knoten antwortet.

Wechselt ein Agent durch eigenes Verhalten oder durch Interaktion mit dem Anwender in den aktiven Zustand, überprüft dieser Agent, ob er bereits seinen zuständigen Kontroll-Knoten kennt. Ist dies nicht der Fall, initiiert er die in Abbildung 3.3 auf der vorherigen Seite abgebildete Kommunikation. Damit versucht sich der neue Agent bei seinem zuständigen Kontroll-Knoten zu registrieren. Für die Registrierung sendet der Agent via Broadcast eine SEARCH-Nachricht an die umliegenden Komponenten. Der zuständige Kontroll-Knoten dieser Gruppe meldet sich daraufhin bei dem Agenten. Von nun an können Überwachungsaufgaben seitens des Kontroll-Knotens erfolgen. Nachfolgend wird beschrieben, wie ein Agent Stromflüsse verhandeln kann.

3.4.2. Beginn einer Verhandlung

Nachdem die Registrierung stattgefunden hat, kann ein Agent mit den Agenten der gleichen Gruppe interagieren. Da der Agent nun seine Nachbarschaft kennt, wird die Kontroll-Schicht lediglich für Kontroll- und Monitoring-Aufgaben benötigt. Damit können Stromflüsse verhandelt werden, selbst wenn der zuständige Kontroll-Knoten ausfällt. Befindet sich ein Agent im aktiven Zustand, kann er gemäß seiner Verhaltensweisen mit seiner Nachbarschaft kommunizieren und den Beginn einer Verhandlung anstoßen. Abbildung 3.4 auf der nächsten Seite zeigt schematisch den Ablauf der Verhandlung bis zum erfolgreichen Abschluss. Eine erfolgreiche Verhandlung besteht aus folgenden den fünf Schritten: (1) REQUEST, (2) PROPOSE, (3) ACCEPT_PROPOSAL, (4) AGREE und (5) INFORM. In der Abbildung benötigt ein Verbraucher Strom. Dazu sendet er via Broadcast ein REQUEST an seine Gruppe. Die Nachricht enthält als Information die benötigte Strommenge. Als Reaktion erhält der Verbraucher ein Versorgungsangebot (PROPOSAL) von der Batterie und dem Erzeuger. Die Reaktion findet nur statt, wenn die Batterie bzw. der Erzeuger genügend Strom zu Verfügung haben. Der Verbraucher entscheidet sich daraufhin für den Erzeuger (ACCEPT_PROPOSAL). Der Grund für diese Entscheidung können Latenzzeiten sein, sodass das Angebot des Erzeugers früher bei dem Verbraucher eintrifft. Weiterhin ist denkbar, dass die Kontroll-Schicht diese Wahl anordnet. Der Erzeuger reagiert mit einem AGREE. Der virtuelle Stromfluss kann beginnen. Da der Verbraucher die initiale Nachricht zum Anstoß der Verhandlung sendet, wird er zum Initiator dieser Verbindung. Abbildung 3.5 auf der nächsten Seite zeigt die Attribute einer Verbindung. Der initiierende Agent setzt die drei Attribute mit den entsprechenden Werten und sendet diese Informationen in dem letzten Schritt mit

einer INFORM-Nachricht an den zuständigen Kontroll-Knoten. Sobald zwei Agenten eine gemeinsame Verbindung aufgebaut haben, kann der virtuelle Stromfluss stattfinden. Der Ablauf während des Stromflusses wird nachfolgendend vorgestellt.

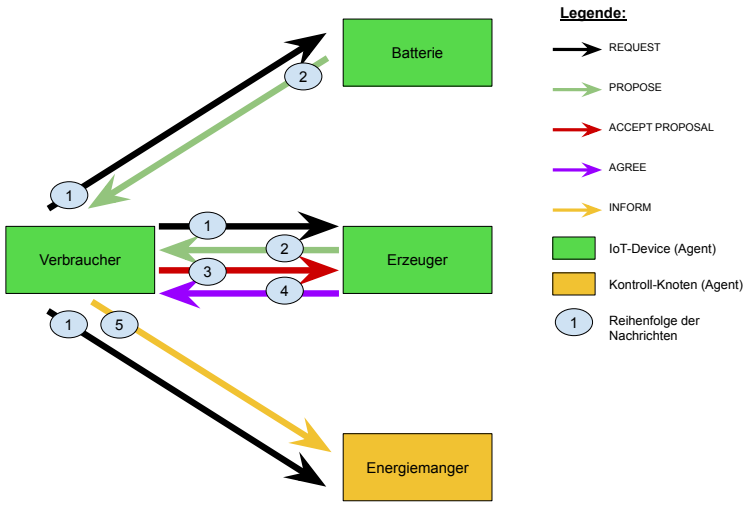


Abbildung 3.4.: Darstellung des Kommunikationsaufbaus. In dem abgebildeten Szenario fragt der Verbraucher innerhalb der Gruppe nach Strom. Der fiktive Stromfluss wird in fünf Schritten aufgebaut. (1) Der Verbraucher sendet ein REQUEST via Broadcast an alle IoT-Devices seiner Gruppe. (2) Die Batterie und der Erzeuger können den Verbraucher versorgen und signalisieren dies durch ein PROPOSE. Der Verbraucher erhält zuerst das Angebot des Erzeugers und sendet ein ACCEPT_PROPOSAL an diesen. Der Erzeuger erstellt intern eine Verbindung und bestätigt dies dem Vebraucher mit einem AGREE. Daraufhin vermerkt auch der Verbraucher eine Verbindung und informiert den Energiemanager bzw. den übergeordneten Kontroll-Knoten. Dieser erhöht den Stromverbrauch der Gruppe um den des Verbrauchers.

Verbindung
• Initiator
• Partner
• Fließende Strommenge

Abbildung 3.5.: Aufbau einer Nachricht. Eine Verbindung hält Informationen über den Initiator der Verhandlung, den weiteren an dem Stromfluss beteiligten Agenten und die virtuell fließende Strommenge. Zum Initiator wird der Agent, der das initiale REQUEST für die jeweilige Verhandlung gesendet hat.

3.4.3. Virtueller Stromfluss

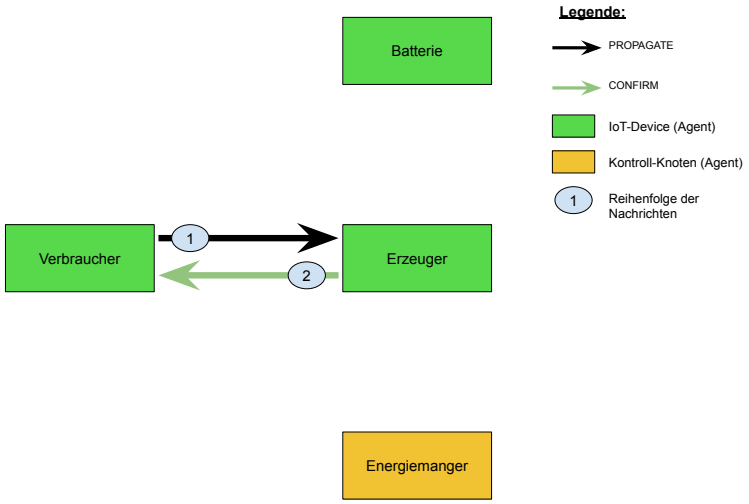


Abbildung 3.6.: Darstellung der Stromversorgung. Die Stromversorgung wird in der Simulation als ein fiktiver Stromfluss über einen Nachrichtenaustausch dargestellt. Die Kommunikation findet nur zwischen dem Verbraucher und dem Erzeuger statt. Da der Verbraucher innerhalb der Gruppe nach Strom gefragt hat, ist er der Initiator der Verbindung. Aus diesem Grund sendet der Verbraucher eine PROPAGATE-Nachricht an den Erzeuger. Der Inhalt der Nachricht ist die vorab verhandelte Strommenge. Der Erzeuger bestätigt den Stromfluss mit einem CONFIRM. Die beiden Nachrichten dienen der Abbildung des virtuellen Stromflusses. Die Abfolge aus PROPAGATE und CONFIRM findet solange statt, wie der verhandelte Stromfluss besteht.

Sobald die Verhandlung erfolgreich abgeschlossen wurde, findet die virtuelle Stromübertragung statt. Die Stromübertragung wird dabei mittels Nachrichtenaustausch visualisiert. Dieser Austausch ist in Abbildung 3.6 dargestellt. In der Abbildung wird ein Verbraucher von einem Erzeuger mit Strom versorgt. Der Verbraucher ist der Initiator der Verbindung. Damit ist es seine Aufgabe, den virtuellen Stromfluss anzustoßen. Der Verbraucher sendet dafür eine PROPAGATE-Nachricht an seinen Versorger. Dieser antwortet mit CONFIRM, um anzuzeigen, dass die Versorgung weiterhin besteht. Der Nachrichtenaustausch findet statt, bis der Verbraucher keinen Strom mehr benötigt, sich dessen Verbrauch erhöht oder der Erzeuger keinen weiteren Strom liefern kann. In den ersten beiden Fällen wird das Übertragungsende durch den Verbraucher eingeleitet, in dem Dritten durch den Erzeuger. Weiterhin ist es möglich, dass die Kontroll-Schicht eingreift oder einer der Kommunikationsteilnehmer nicht mehr antwortet. Dann wird ebenfalls die Übertragung beendet. Das Übertragungsende thematisiert der folgende Unterabschnitt.

3.4.4. Übertragungsende

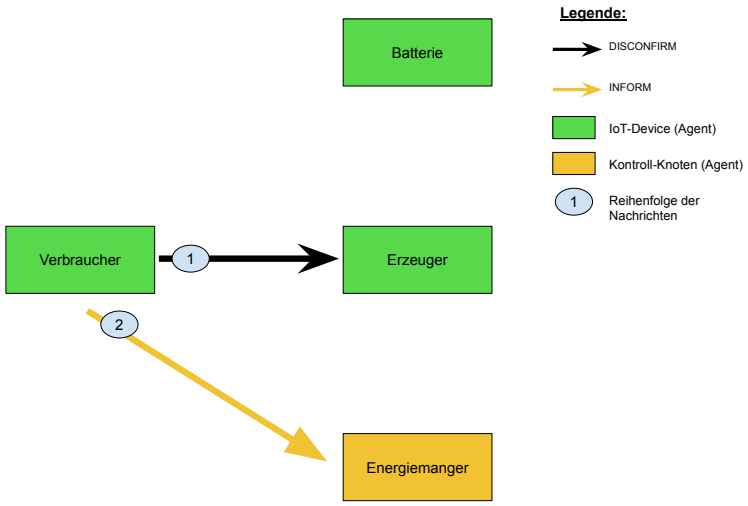


Abbildung 3.7.: Darstellung des Kommunikationsabbaus. In der Abbildung beendet der Verbraucher den virtuellen Stromfluss und leitet das Übertragungsende ein. Eine mögliche Ursache ist, dass der Verbraucher nicht weiter mit Strom versorgt werden muss. Für den Abbau der Verbindung sendet dieser an seinen Versorger eine DISCONFIRM-Nachricht und löscht intern die Verbindung. An den Energiemanager bzw. an den übergeordneten Kontroll-Knoten wird eine INFORM-Nachricht gesendet. Der Erzeuger schließt nach Erhalt des DISCONFIRM ebenfalls die Verbindung und ist nun wieder bereit, andere IoT-Devices zu versorgen. Der übergeordnete Kontroll-Knoten löscht ebenfalls die Verbindung und vermerkt, dass nun weniger Strom verbraucht wird.

Damit das Übertragungsende geregelt abläuft, gibt es auch hierfür ein nachrichtenbasiertes Protokoll. Ein Übertragungsende tritt auf, wenn kein weiterer Strom benötigt wird, eine Versorgung nicht aufrecht erhalten werden kann, die Kontroll-Schicht eingreift oder einer der Teilnehmer ausgeschaltet wird bzw. ausfällt. In Abbildung 3.7 ist dieses Übertragungsende illustriert. Der abbrechende Agent sendet eine DISCONFIRM-Nachricht an seinen Gegenüber und informiert den übergeordneten Kontroll-Knoten mit einer INFORM-Nachricht über den nicht länger bestehenden Stromfluss. Der Kontroll-Knoten passt daraufhin Energieverbrauch und -erzeugung der untergeordneten Gruppe an. Die Kommunikation und somit der virtuelle Stromfluss sind beendet.

Die Phasen des Kommunikationsprotokoll sowie das in Abschnitt 3.3 auf Seite 34 präsentierte agentenbasierte Modell müssen nun in ein ganzheitliches, den definierten

Anforderungen weitgehend entsprechendes Konzept gefasst werden. Dazu werden in dem nachfolgenden Abschnitt die Architektur des Simulator-Konzepts vorgestellt und Alternativen abgewogen.

3.5. Architektur des Simulators

In diesem Abschnitt werden die identifizierten Komponenten eines Electric-Grid Ökosystems auf eine geeignete Software-Architektur abgebildet. Die Architektur strukturiert dabei das agentenbasierte Modell und das Kommunikationsprotokoll in eine Form, welche im weiteren Verlauf für die prototypische Implementierung zur Hand genommen wird. Bei der Wahl der Architektur ist zu beachten, dass die in Abschnitt 3.2 auf Seite 26 definierten Anforderungen unterstützt und Alternativen gegeneinander abgewogen werden.

In der Szenarien-Anlayse und der Entwicklung eines agentenbasierten Modells sind insbesondere die Gemeinsamkeiten in dem Ablauf und Aufbau der Szenarien sowie der darin verwendeten Agenten und deren Verhalten aufgefallen. Aufgrund der herausgestellten Gemeinsamkeiten wurde für die Architektur des Simulators auf die Konzepte der objektorientierten Programmierung zurückgegriffen und insbesondere das Prinzip der Vererbung genutzt. Für ein hierarisches Konzept spricht zudem, dass die Agenten sich nicht in allen Eigenschaften ähneln und über spezialisierte Klassen die Unterschiede abgebildet werden können.

Abbildung 3.8 auf der nächsten Seite zeigt schematisch die resultierende Hierarchie der Agenten. Die Wurzel dieser Hierarchie bildet der *Electric-Grid-Agent*. Der *Electric-Grid-Agent* bildet die grundlegenden Eigenschaften und das Verhalten eines Agenten in einer Electric-Grid Umgebung ab. Zu den Eigenschaften, die jeder dieser Agenten besitzt, gehören nach Abschnitt 3.3 auf Seite 34 insbesondere eine eindeutige ID sowie die Möglichkeit zur Interaktion bzw. Kommunikation mit anderen umliegenden Agenten. Die vier Kategorien von IoT-Devices bringen jeweils individuelle Charakteristika ein. Ein Verbraucher hat beispielsweise einen Verbrauch, ein Erzeuger eine verfügbare Strommenge, ein Energiespeicher eine Kapazität und ein Hausanschluss einen variablen Strompreis. Eine Übersicht über alle Attribute findet sich in Tabelle 3.1 auf Seite 36. Die drei Kontroll-Device Kategorien *Oberste Kontroll-Schicht*, *SNN* und *Energiemanager* unterscheiden sich ebenfalls von einem *Electric-Grid-Agenten* und erweitern diesen im Wesentlichen um die Kontroll- und Monitoring-Aufgabe. Eine vollständige Übersicht über die Attribute der Kontroll-Schichten ist in Tabelle 3.2 auf Seite 37 abgebildet. Da die Regeln für die Kontroll-Tätigkeiten in dieser Arbeit nicht spezifiziert werden und die Agenten sich in dem Monitoring-Verhalten nicht signifikant unterscheiden, werden die drei Kategorien in der Architektur in einem Kontroll-Knoten zusammengefasst.

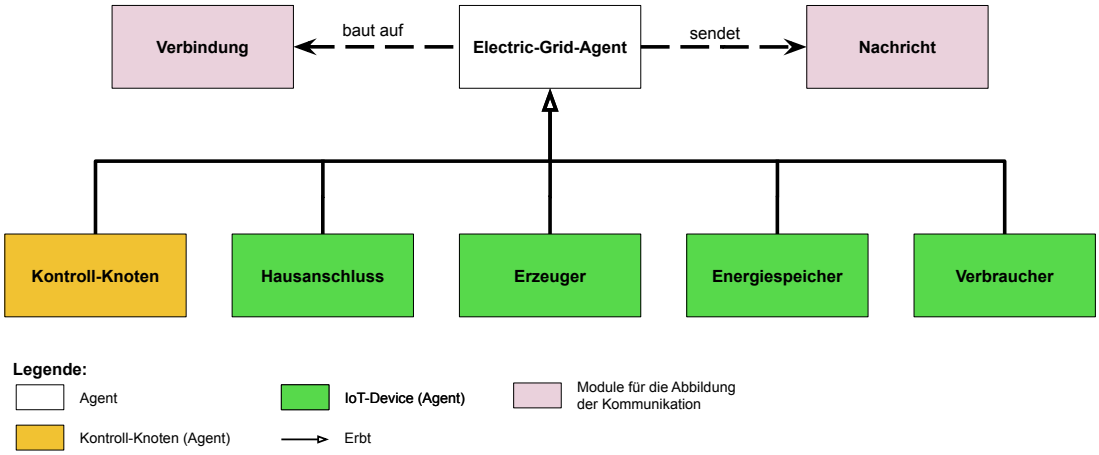


Abbildung 3.8.: Architektur des Simulators. Die Architektur des Szenarien-Simulators folgt der aus der Objektorientierung bekannten hierarchischen Vererbung. Der Electric-Grid-Agent bildet das Wurzelelement der Hierarchie und enthält die grundlegenden Attribute und Verhaltensweisen eines Agenten in einer Electric-Grid Umgebung, wie etwa eine eindeutige ID und die Möglichkeit zur Kommunikation. Von diesem Wurzelelement erben alle weiteren IoT-Devices und Kontroll-Knoten. Die Vererbung ist über die schwarze durchgezogene Linie abgebildet. Ein Agent ist dabei einer der fünf Kategorien Kontroll-Knoten, Hausanschluss, Erzeuger, Energiespeicher oder Verbraucher zuordbar. In seinem Grundverhalten kann ein Agent Nachrichten versenden und Stromflüsse verhandeln. Weiterhin kann er bei erfolgreicher Verhandlung eine Verbindung zu einem zweiten Agenten aufbauen.

Zudem sind in Abbildung 3.8 mit dem Nachrichten- und Verbindungs-Modul die benötigten Module für die Kommunikation abgebildet. Instanzen des Nachrichten-Moduls werden während der Kommunikation von den Agenten ausgetauscht. Eine einzelne Nachricht beinhaltet nach Abbildung 3.2 auf Seite 43 unter anderem Informationen über den Sender, den Empfänger und die Kommunikationsabsicht. Nach erfolgreicher Verhandlung bauen die Agenten eine Verbindung auf, um den virtuellen Stromfluss darzustellen. Die Verbindung enthält nach Abbildung 3.5 auf Seite 46 Daten über die Kommunikationsteilnehmer und die virtuell fließende Strommenge.

Eine Alternative zu dem gewählten klassenhierarchischen Konzept stellt die Verwendung und Konfiguration eines Dummy-Agenten über Konfigurationsfiles dar. Ein solcher Ansatz ist vergleichbar mit der Architektur der DIMMER Plattform aus [KJP15]. Die Idee hinter der Architektur ist, dass ein einziges Klassengerüst für alle Agentenkategorien spezifiziert wird. Die Instanzen dieser Klasse haben während des Betriebs die Möglichkeit, über definierte Schnittstellen Regeln und Verhaltensweisen aus einer Datenbank, Cloud, etc. zu laden. Diese geladenen Konfigurationsfiles

ermöglichen ein individuelles Verhalten des Dummy-Agenten und damit eine Abbildung auf die in Abschnitt 3.3 auf Seite 34 spezifizierten IoT-Devices und Kontroll-Knoten. Ein Dummy-Agent nimmt in dieser Art von Architektur somit während des Betriebs eine Rolle ein. Da ein Dummy-Agent die Regeln und Verhaltensweisen nur lädt und diese nicht fest in der Agentenkategorie verankert sind, ist es möglich, dass Agenten während der laufenden Simulation ihre Rollen wechseln oder aktualisierte Regeln und Verhalten der alten Rolle laden. Mit der Architektur wird die Neukonfiguration der eingesetzten Agenten und damit Veränderungen innerhalb des simulierten Szenarios vereinfacht, ohne bei jeder Änderung neue Agenteninstanzen erstellen zu müssen. Diese Eigenschaft bietet einen Vorteil gegenüber der ausgewählten Klassenhierarchie, in der zwar auch Änderungen der Agentenrollen während des Betriebs erfolgen können, diese jedoch in der Neuerstellung von Instanzen resultieren. Die Änderungen von Agentenparametern wie etwa die aktuelle Erzeugung oder der Verbrauch sind in beiden Architekturen über entsprechende Methoden abbildbar. Einen Nachteil, den die alternative Architektur beinhaltet, ist die Erweiterbarkeit des Konzepts. Die Architektur setzt voraus, dass jeder Agent gleich konfigurierbar ist und über die selben Schnittstellen agiert und erreichbar ist. Dazu gehört auch, dass die Agenten jeweils über die gleiche Anzahl und die gleiche Bezeichnung der Attribute und Methoden verfügen. Wird ein File eines neuen Agenten hinzugefügt, muss die Konfiguration exakt gleich verlaufen oder zieht Änderungen in der Basisklasse mit sich. Dies führt dazu, dass einige Agenten nicht benötigte Attribute, Methoden und Schnittstellen besitzen. Die gewählte Klassenhierarchie umgeht daraus resultierende Probleme, beispielsweise, dass ein Zugriff auf falsche Attribute erfolgt, indem ein neu hinzugefügter Agent in einer separaten Klasse abgebildet wird. Somit verfügt jeder Agent nur über die benötigten Attribute und gemeinsames Verhalten kann in Oberklassen gehalten werden. Damit unterstützt die gewählte Architektur Erweiterbarkeit (Anforderung (Q3)) und Domänenunabhängigkeit (Anforderung (Q4)).

In diesem Kapitel wurde die schrittweise Entwicklung des theoretischen Simulator-Konzepts vorgestellt. Dazu wurden zu Beginn des Kapitels die Szenarien bezüglich ihrer Gemeinsamkeiten und Unterschiede untersucht. Als eine wichtige Gemeinsamkeit wurden dabei die verwendeten Agenten identifiziert. Aus der Analyse wurden funktionale und qualitative Anforderungen an den späteren Simulator definiert. Im Weiteren wurde die ABMS angewendet und ein agentenbasiertes Modell eines Electric-Grid erstellt. Als eine wesentliche Eigenschaft eines Agenten wurde dabei die Kommunikation mit anderen Agenten herausgestellt. Damit die Agenten eine gemeinsame Sprache sprechen, wurde anschließend das entwickelte Kommunikationsprotokoll erläutert. Zum Schluss wurden die über das Kapitel identifizierten Komponenten auf eine geeignete Architektur abgebildet. In dem nachfolgenden Kapitel wird eine mögliche Realisierung des vorgestellten Konzepts beschrieben. Dabei werden unter anderem die Wahl der Programmiersprache und technische Aspekte der Realisierung besprochen.

4. Realisierung des Simulators

Dieses Kapitel beschreibt die praktische Realisierung des Szenarien-Simulators. Wie in der Einleitung beschrieben, wird das erstellte Konzept prototypisch umgesetzt, damit im späteren Verlauf des Projekts Ansätze der Kontroll-Schicht getestet werden können. Das Kapitel gliedert sich in zwei Abschnitte. Zu Beginn des Kapitels wird in dem ersten Abschnitt die Wahl der Vorgehensweise und der Programmiersprache begründet. Zur Auswahl standen dabei diverse Simulationstools, existierende Simulatoren und objektorientierte Programmiersprachen. Der zweite Abschnitt befasst sich mit der technischen Umsetzung des theoretischen Konzepts. Anlehnend an das Konzept müssen die Agenten in der Lage sein, mittels Nachrichten zu kommunizieren, um Absprachen bezüglich der Stromflüsse zu treffen. Dazu wird in diesem Abschnitt unter anderem die Realisierung der Kommunikation erläutert. Weiterhin wird eine Übersicht über die implementierte GUI gegeben und die Bedienung dargelegt. Zum Ende des Abschnitts wird die Einbindung und die Überwachungsaufgabe der Kontroll-Schicht präsentiert.

4.1. Vorgehensweise und Programmiersprache

In diesem Abschnitt wird die Vorgehensweise zur Erstellung eines prototypischen Simulators beschrieben. Da das theoretische Konzept mittels der ABMS entworfen wurde, ist eine Anforderung an die Vorgehensweise, dass Agenten und deren Verhaltensweisen spezifiziert werden können. Für die Implementierung eines agentenbasierten Modells kann nach Macal und North [MN10] prinzipiell eine beliebige höhere Programmiersprache oder ein speziell entwickeltes Toolkit verwendet werden. In [MN08] werden mit *NetLogo*, *Repast* und *Swarm* einige dieser speziellen Toolkits vorgestellt. Die Toolkits bieten insgesamt ein breites Spektrum an Anwendungsmöglichkeiten, sind jedoch in ihrem Funktionsumfang, beispielsweise bei der Integration neuer Protokolle, eingeschränkt. So können etwa bei *Repast* Agenten und die Struktur des Modells spezifiziert werden, der Fokus liegt dann jedoch auf der Visualisierung der Agentenaktionen und nicht auf der internen Realisierung [MN08]. Um während der Umsetzung des Simulators nicht auf den Funktionsumfang eines Toolkits angewiesen zu sein und damit der realisierte Prototyp im weiteren Verlauf individuell erweiterbar bleibt, fiel die Entscheidung auf die Nutzung einer Programmiersprache. Die objektorientierten Sprachen bieten sich dabei besonders an, da ein

Objekt mit seinen Attributen und Methoden vergleichbar mit einem Agenten und dessen Wissen und Verhaltensweisen ist [WR17].

Als nächstes stellte sich die Frage, ob ein in einer objektorientierten Sprache entwickelter Simulator erweitert oder der Prototyp von Grund auf neu implementiert werden soll. Die Analyse und anschließende Erweiterung eines ausgereiften Simulatorkonzepts bzw. eines lauffähigen Simulators, welche im Kontext eines Electric-Grids oder anderen IoT-Szenarien erstellt wurden, bieten den Vorteil, dass in einer frühen Phase der Arbeit bereits Schwierigkeiten und Probleme sichtbar werden. Die Erweiterung bestehender Ansätze kann den Implementierungsaufwand der Grundstrukturen reduzieren und mit der gewonnen Zeit kann das eigene Simulator-Konzept verfeinert werden. In Abschnitt 1.1 auf Seite 2 ist ein Überblick über einige existierende Ansätze gegeben. Dort wurde bereits erwähnt, dass eine Vielzahl der Ansätze nicht ausreichend dokumentiert ist und die Quellcodes häufig nicht verfügbar sind. Hinzu kommt, dass die Fokusse existierender Ansätze von den Zielen dieser Arbeit abweichen. Aus diesen Gründen wurden die existierenden Konzepte zwar analysiert, jedoch auf eine Erweiterung verzichtet.

Bei der Recherche nach existierenden Simulatoren war auffällig, dass diese in der Regel in einer der beiden Programmiersprachen *Java* und *Python* realisiert sind. Für beide der Sprachen ist ein Entwicklungsframework für agentenbasierte Systeme vorhanden. Das Java Agent Development (JADE) Framework [BCPR03] und das Python Agent Development (PADE) Framework [MSL+19] bieten ein breites Leistungsspektrum an. Dies beinhaltet auch das Versenden von Daten und die Reaktion auf eingehende Nachrichten. Somit ist die Grundvoraussetzung für die Anwendung erfüllt. Für das Framework PADE sprach die Aktualität (2019) sowie die Einbindung der FIPA Protokolle, insbesondere des in Abschnitt 3.4 auf Seite 42 erwähnten ACL-Protokolls. Aus diesem Grund fiel die Wahl der Programmiersprache auf Python. Während der Implementierung mittels PADE sind Probleme bei Kommunikation zwischen den Agenten aufgetreten. Da die Kommunikation eine essentielle Verhaltensweise der Agenten ist und das Tool keine wesentlichen weiteren Vorteile bot, wurden die implementierten Ansätze verworfen. Der nachfolgende Abschnitt greift diese und andere Schwierigkeiten während der Implementierungsphase auf und beschreibt die technische Umsetzung in den beiden realisierten Prototypen.

4.2. Technische Umsetzung

Dieser Abschnitt beschreibt die technische Übersetzung des theoretischen Konzepts in einen prototypischen Simulator und ist wie folgt gegliedert: Zu Beginn erläutert der erste Unterabschnitt, wie die spezifischen Charakteristika und Verhaltensweisen eines Agenten in Klassen abgebildet wurden. Bei der Implementierung wurde

insbesondere versucht, autonome Agenten zu erhalten und die Anforderungen aus Abschnitt 3.2 auf Seite 26 zu erfüllen. Der zweite Unterabschnitt stellt detailliert die Umsetzung der Kommunikation zwischen den Agenten dar. Die Agenten müssen in der Lage sein, Daten miteinander auszutauschen und auf eingehende Nachrichten zu reagieren. Der dritte Unterabschnitt beschreibt die Simulations-Klasse. Diese Klasse ist für den Aufbau der Szenarien und die Koordination während der laufenden Simulation zuständig. Der vierte und letzte Unterabschnitt thematisiert die Einbindung und Darstellung einer Kontroll-Schicht. Die Kontroll-Schicht hat im jetzigen Zustand nur eine Überwachungsaufgabe. Weitere Regeln zur Ausübung von Kontrolle werden erst später implementiert. Auf die Umsetzung der Nachrichten-, Verbindungs- sowie einiger Hilfsklassen wird an dieser Stelle nicht näher eingegangen, da die Klassen lediglich die in Abschnitt 3.4 auf Seite 42 vorgestellten Attribute enthalten bzw. im Fall der Hilfsklassen übersichtliche Enums abbilden.

4.2.1. Agentenverhalten und Realisierung der Kommunikation

An dieser Stelle wird beschrieben, wie die Agenten und insbesondere das Agentenverhalten aus Unterabschnitt 3.3.4 auf Seite 39 in Klassen eines Simulators umgesetzt wurden. In der Realisierungsphase sind zwei aufeinander aufbauende Prototypen entstanden, die zwar in dem implementierten Verhalten identisch sind, sich jedoch in den Verhaltensweisen während der Simulation aufgrund verschiedener Ablaufsteuerungen bei den Interaktionen unterscheiden.

Mit der ABMS wurden insgesamt fünf verschiedene Arten von smarten Komponenten identifiziert: (1) Kontroll-Knoten, (2) Hausanschluss, (3) Erzeuger, (4) Energiespeicher und (5) Vebraucher. Da diese fünf Arten überlappende Eigenschaften aufweisen, wurden die gemeinsamen Attribute und Verhaltensweisen in einem übergeordneten Modul, der *Electric-Grid-Agent*-Klasse, abgebildet. Zu den gemeinsamen Attributen gehören unter anderem eine numerische, eindeutige Agenten-ID und ein binärer Aktivitätsstatus. Spezielle Attribute (z.B. der Verbrauch eines Agenten) sind in den erweiternden Klassen (z.B. Verbraucher) über Attribute des entsprechenden Typs abgebildet. Beispiele für gemeinsame Verhaltensweisen sind die Möglichkeit zur Reaktion auf eingehende Nachrichten und das Versenden von Daten. Die Agenten reagieren jeweils individuell auf eingehende Nachrichten. Bei der Reaktion der Agenten spielt der in der Nachricht spezifizierte Kommunikationsgrund (z.B. REQUEST) eine wichtige Rolle. Dafür hat jeder Agent entsprechende *react_<Kommunikationsgrund>*-Methoden. In den *react-* und *send*-Methoden wurde jeweils das Verhalten aus Unterabschnitt 3.3.4 auf Seite 39 implementiert. Im Anhang ist unter Abschnitt A.2 auf Seite 84 das ausführliche Klassendiagramm der Simulatoren abgebildet. Die Bezeichnungen der Klassen, Methoden und Attribute

sind bei beiden Prototypen identisch, sodass das Klassendiagramm sowohl für die erste, als auch für die zweite prototypische Implementierung gültig ist.

Im Wesentlichen unterscheiden sich die zwei prototypischen Implementierungen in der Simulations-Klasse. Für den ersten Prototypen wird in dieser Klasse eine Liste der Agenten eines Szenarios erstellt. Diese Liste wird nach dem Start des Simulators iterativ in einer Schleife durchlaufen. Ist ein Agent an der Reihe, prüft er, ob er sich im aktiven oder inaktiven Zustand befindet. Im inaktiven Zustand wird dieser Agent übersprungen. Im aktiven Zustand führt der Agent eine der Situation entsprechende Aktion aus. Nach Abschluss der Aktion ist der nächste Agent der Liste an der Reihe. Dies impliziert, dass eine vorgeschriebene Reihenfolge bezüglich der Agenten besteht. Damit Agenten trotzdem eingehende Nachrichten erhalten, beinhaltet jeder Agent einen Puffer, in dem in der Zwischenzeit ankommende Nachrichten zwischengespeichert werden. Da reale IoT-Devices eigenständig handeln können und nicht an Reihenfolgen gebunden sind, wurde bei der Entwicklung des zweiten Prototypen versucht, diese Eigenständigkeit zu erreichen. Dazu wurde anstelle einer Liste für jeden Agenten ein eigener Thread erstellt. Werden die Threads gestartet, läuft jeder einzelne Agent in einer separaten Schleife. Diese parallele Abarbeitung hat einerseits den Vorteil, dass die in der Realität separaten IoT-Devices getreuer auf Software abgebildet werden. Andererseits bestehen weniger Abhängigkeiten zwischen den Agenten, wodurch diese autonomer handeln können. Da während der Simulation einige Attribute von mehren Agenten und Modulen verwendet werden, wurden zudem entsprechende Locks eingerichtet. Diese Locks sperren die jeweiligen Attribute, um einen exklusiven Zugriff zu erhalten. Dadurch werden potenzielle Lese-Schreib-Anomalien vermieden.

Für die Umsetzung der Kommunikation war vorerst beabsichtigt, das PADE Framework zu verwenden. Da insbesondere in dieser Funktionalität Fehler in der Reaktion auf eingehende Daten auftraten, musste eine Alternative gefunden werden. Eine Möglichkeit bestand darin, ein Peer-to-Peer Package der Sprache Python zu verwenden. Die Recherche hat jedoch ergeben, dass sich die Verwendung dieser Packages besonders in größeren Systemen als komplex und unübersichtlich gestaltet. Des Weiteren stand die Verwendung eines zentralen Servers und die Abbildung der Agenten auf Clients zur Auswahl. Der Datenaustausch der Agenten-Clients würde in diesem Fall über den zentralen Server erfolgen, was entgegen des verteilten Ansatzes des IoTs ist. Die dritte Alternative war der gegenseitige Aufruf von Methoden. Dies widerspricht jedoch dem Gedanken der Netzwerkkommunikation. Im Hinblick auf die Bearbeitungszeit stach die Verwendung von nicht-blockierenden Sockets als relativ einfach und schnell umzusetzende Alternative heraus. In jedem Agent wird dazu ein eigener Socket mit eindeutiger Adresse als Attribut gehalten. Für die direkte Kommunikation benötigt jeder Agent die Socket-Adressen der Gruppenmitglieder und des zugeordneten Kontroll-Knotens. Durch die direkte Kommunikation geht dieser Ansatz in Richtung eines Peer-to-Peer-Netzwerkes. Die Sockets wurden mit als nicht-

blockierend implementiert, damit die Agenten während des Betriebs nicht an den entsprechenden Programmstellen verharren und auf eingehende Nachrichten warten müssen, sondern mit ihren Verhaltensweisen und Aktionen fortfahren können. Als Netzwerkstandard wurde das User Datagram Protocol (UDP)[1] gewählt. UDP bietet im Vergleich zu dem Transmission Control Protocol (TCP)[2] einen verbindungslosen Datenaustausch. Dies hat den Vorteil einer schnelleren und effizienteren Kommunikation, da keine Netzwerkverbindung auf- und abgebaut werden muss. Wie die Kommunikation und damit das Agentenverhalten über die GUI visualisiert und intern koordiniert wird, thematisiert der nächste Unterabschnitt.

4.2.2. GUI und Simulations-Klasse

In diesem Unterabschnitt wird der Aufbau und die Funktionalitäten der GUI sowie die Umsetzung der Simulations-Klasse, die die Erstellung und den Ablauf der Szenarien koordiniert, vorgestellt. Dazu orientiert sich dieser Unterabschnitt an den während der Erstellung und Ausführung einer Simulation ablaufenden Schritte. Die drei Schritte sind: (1) Programmstart, (2) Szenarioerstellung und (3) Simulation.

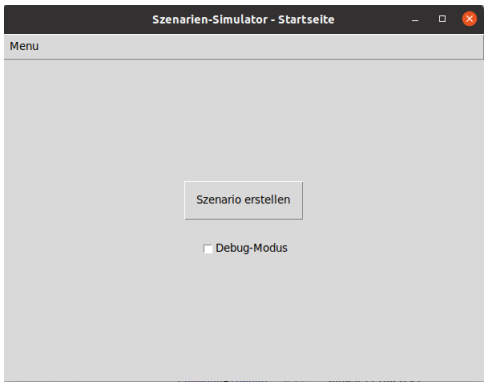


Abbildung 4.1.: Screenshot des Programmstarts. Die Abbildung zeigt das Startfenster der Simulatoren. Zu Beginn der Simulation hat ein Anwender die Möglichkeit, den Debug-Modus auszuwählen. Ist dieser Modus ausgewählt, werden die Agentenaktionen sowohl temporär über die GUI bereitgestellt, als auch in einer ausführlicheren Version in eine separate Datei geschrieben. Diese Datei kann auch noch nach Abschluss der Simulation eingesehen und analysiert werden. Der Debug-Modus soll zum einen dabei unterstützen, etwaige Fehler des Simulators aufzudecken und zum anderen zu einem tieferen Verständnis des Modells und potenzieller Emergenzen beitragen.

[1] Siehe: https://tools.ietf.org/html/rfc768
[2] Siehe: https://tools.ietf.org/html/rfc793

In Abbildung 4.1 auf der vorherigen Seite ist der erste Schritt dargestellt. Nach dem Start der Software kann der Anwender zwischen Normalem- und Debug-Modus wählen. Den Debug-Modus unterscheidet dabei, dass die Agenteninteraktionen (ausgetauschte Nachrichten) sowohl temporär in der GUI visualisiert, sowie in einer separaten Datei gespeichert werden. Die Datei ist auch nach Beenden des Simulators einsehbar. Die in der Datei dargestellten Interaktionen enthalten zudem weiterführende Informationen über den jeweiligen Agententyp der kommunizierenden Agenten sowie die zu verhandelnde bzw. von den Agenten bereits verhandelte Strommenge. Mit dieser Datei kann der Simulator auf Fehler überprüft und das Szenario auf Emergenzen untersucht werden.

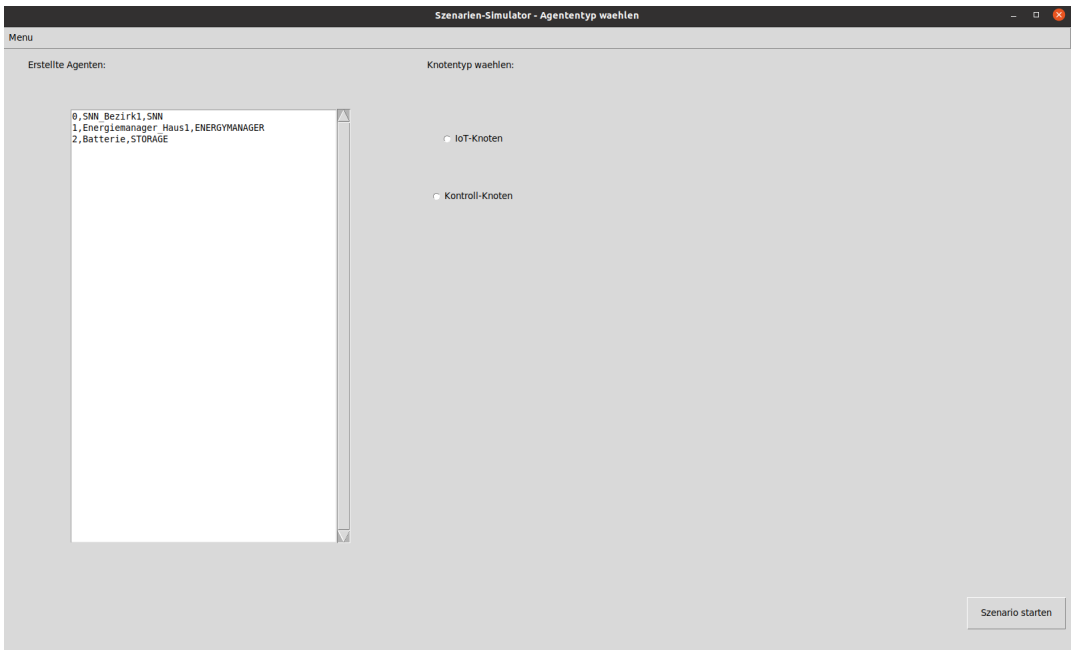


Abbildung 4.2.: Screenshot der Szenarienerstellung. In der Abbildung ist das Fenster für die Erstellung von Electric-Grid Szenarien dargestellt. In dem Textfeld auf der linken Seite sieht der Anwender alle bisher erstellten Agenten. In dem aktuellen Szenario wurden bereits ein SNN, ein Energiemanager und ein Energiespeicher erstellt. Für die Konfiguration weiterer Agenten kann der Anwender zwischen IoT- und Kontroll-Knoten wählen. Durch die Auswahl *IoT-Knoten* werden dem Anwender die Kategorien *Verbraucher*, *Erzeuger*, *Energiespeicher* und *Hausanschluss* angezeigt. Nach Auswahl einer der Kategorien kann der Anwender individuelle Parameter (z.B. übergeordneten Agenten oder den Verbrauch) spezifizieren und einen Agenten mit diesen Eigenschaften erstellen. Analog stehen im Fall der Kontroll-Knoten die Kategorien *Energiemanager* und *SNN* zur Auswahl. Wurde mindestens ein Agent erstellt, kann die Simulation gestartet werden. Für den Start ist dabei nicht von Bedeutung, welcher Kategorie dieser Agent entstammt.

In dem zweiten Schritt kann der Anwender ein Szenario erstellen. Dazu wählt er nacheinander Agenten aus den Kategorien IoT- und Kontroll-Knoten aus und spezifiziert diese über weitere Dialoge. Damit ist es möglich, beliebige Agenten der Kategorien *Verbaucher*, *Erzeuger*, *Energiespeicher*, *Hausanschluss*, *SNN* und *Energiemanager* zu erstellen. Die Erstellung einer Instanz der obersten Kontroll-Schicht ist in dieser Version des Simulators noch nicht gegeben. Abbildung 4.2 auf der vorherigen Seite zeigt einen Screenshot der Szenarienerstellung. In dem abgebildeten Szenario wurden bereits drei verschiedene Agenten spezifiziert. Die erstellten Agenten werden dem Anwender während des Erstellungsprozesses angezeigt. Wurde mindestens ein Agent, unabhängig vom Typ, erstellt, kann ein Szenario gestartet werden. Bei dem Start des Szenarios kommt die Simulations-Klasse bzw. eine Instanz dieser Klasse ins Spiel. Für jedes Szenario existiert genau eine Instanz dieser Klasse. In dieser Instanz wird die Hierarchie zwischen den Agenten aufgebaut. Im Zuge dessen wird den Kontroll-Knoten ihre untergeordnete Gruppe zugeteilt. Im Anschluss an diese Zuteilung ist es die Aufgabe der Kontroll-Knoten, den ihnen direkt untergeordneten Agenten die jeweiligen möglichen Kommunikationspartner mitzuteilen. Folglich kann in der späteren Simulation nur eine Kommunikation zwischen benachbarten Agenten erfolgen, wenn der ihnen übergeordnete Kontroll-Knoten spezifiziert ist. Diese Umsetzung soll die Registrierung der Agenten abbilden. In der Simulations-Instanz wird zudem die Liste der Agenten verwaltet und die Schleife bzw. die separaten Threads erstellt.

Der dritte und letzte Schritt ist die Simulation des erstellten Szenarios. Abbildung 4.3 auf der nächsten Seite zeigt eine laufende Simulation mit vier Agenten. In der Abbildung ist die Hierarchie der beteiligten Agenten dargelegt. Der SNN kontrolliert den Energiemanager, welcher wiederum die Batterie und den Fernseher überwacht. Die Zugehörigkeiten zur IoT- und Kontroll-Schicht sind jeweils über eine farbliche Markierung gekennzeichnet (grün für IoT-Schicht, orange für Kontroll-Schicht). Da in dem Szenario die jeweiligen übergeordneten Knoten gesetzt wurden, kann Kommunikation zwischen den Agenten stattfinden. Dadurch ist es möglich, dass der Fernseher und die Batterie einen gemeinsamen Stromfluss verhandeln, bei dem die Batterie den Fernseher versorgt. Die Batterie unterbricht nach kurzer Zeit den Stromfluss (DISCONFIRM). Der Fernseher sucht daraufhin aktiv nach einem neuen Versorger (REQUEST). Während der Simulation hat der Anwender die Möglichkeit, die Agentenparameter einzusehen und zu verändern. Zu diesen Parametern gehören je nach Agent beispielsweise der binäre Aktivitätsstatus, der momentane Verbrauch oder die Kapazität eines Speichers. Mit einem Klick auf *Speichern* werden die Änderungen in das Szenario übernommen. Die Koordination dieser Übernahme ist ebenfalls Aufgabe der Simulations-Instanz. Wie in Unterabschnitt 4.2.1 auf Seite 55 beschrieben wurde, müssen für den zweiten Prototypen aufgrund der parallel laufenden Agenten entsprechende Sperren für die Attribute des geänderten Agenten gesetzt werden. Über den Button *Simulation beenden* wird die laufende Simulation beendet.

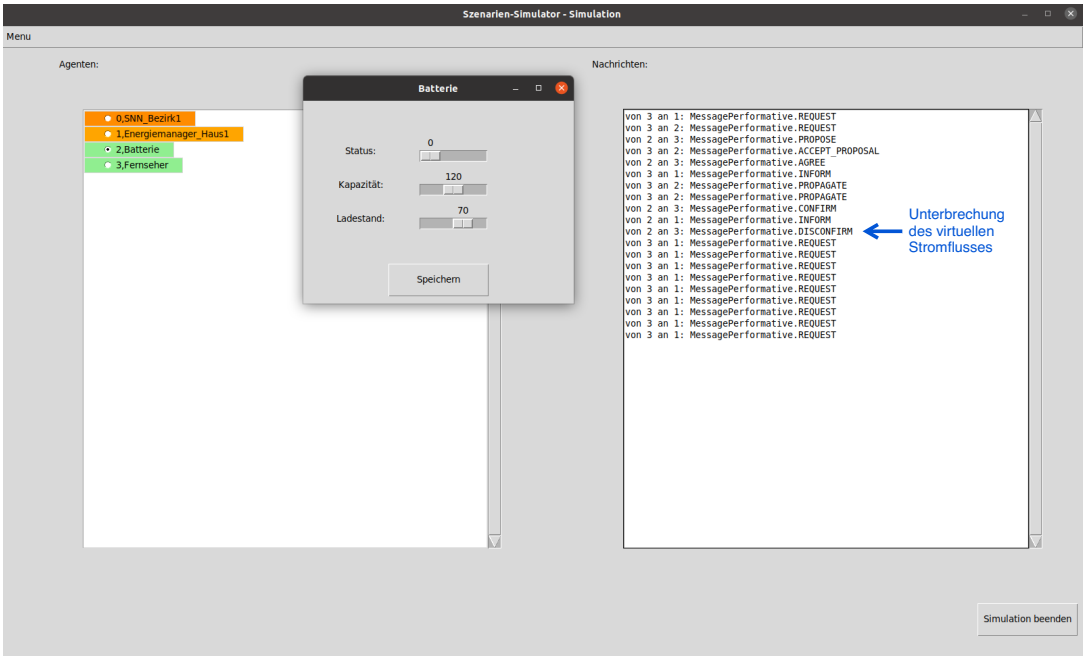


Abbildung 4.3.: Screenshot der laufenden Simulation. Die Abbildung zeigt die laufende Simulation. In dem linken Feld sind die Agenten des Szenarios dargestellt. In dem simulierten Szenario existieren vier Agenten, die entsprechend ihrer Zugehörigkeit farblich gekennzeichnet sind. Die Batterie und der Fernseher gehören zu der IoT-Schicht und sind aus diesem Grund grün markiert. SNN und Energiemanager sind entsprechend der Kontroll-Schicht orange. Um zu kennzeichnen, dass ein SNN auf einer anderen (höheren) Ebene als der Energiemanager agiert, wurde ein anderer Orangeton gewählt. In dem rechten Feld sind die Interaktionen der Agenten visualisiert. Die ausgetauschten Nachrichten zeigen, dass eine Verbindung zwischen dem Fernseher (3) und der Batterie (2) hergestellt und er Energiemanager (1) über diesen Stromfluss informiert wurde. Nach einem kurzen Stormfluss (PROPAGATE-CONFIRM) wurde die Verbindung durch die Batterie (2) mit einem DISCONFIRM beendet, da die Batterie in den inaktiven Zustand versetzt wurde. Diese Aktion kann durch die Auswahl eines Agenten während der Simulation werden geschehen. Dem Anwender werden dann die veränderbaren Parameter angezeigt. Für die Batterie (Energiespeicher) sind dies der Aktivitätsstatus, die Kapazität und der Ladestand. Über die Schieberegler können die Werte entsprechend angepasst werden. Nach der Stromunterbrechung wird der Fernseher nicht mehr versorgt, weshalb dieser wieder aktiv nach möglichen Versorgern sucht.

In dem nachfolgenden Unterabschnitt wird beschrieben, wie die Kontroll-Schicht zum jetzigen Zeitpunkt während einer laufenden Simulation ihrer Überwachungs-Aufgabe nachgeht und welche Vorkehrungen bereits getroffen wurden, um einen späteren Eingriff der Kontroll-Knoten umzusetzen.

4.2.3. Einbindung einer Kontroll-Schicht

In diesem Unterabschnitt wird die Einbindung und Darstellung der Kontroll-Schicht beschrieben. Unterabschnitt 3.3.4 auf Seite 39 hat bereits dargelegt, dass die Aufgabe der Kontroll-Schicht in den Prototypen vorerst in der Überwachung der Stromflüsse besteht. Mit diesen Informationen können die Kontroll-Knoten später nach definierten Regeln in die Interaktionen ihrer untergeordneten Agenten eingreifen.

Durch die Interaktionen ihrer untergeordneten Agenten erhalten die Kontroll-Knoten REQUEST- und INFORM-Nachrichten. Die REQUEST-Nachrichten zeigen an, dass die Agenten in dem Zuständigkeitsbereich Strom abgeben können oder benötigen. Die Nachrichten dieses Typs können im späteren Verlauf ein Anstoß für Eingriffe der Kontroll-Schichten sein. Da über die GUI bereits Interaktionen mit den Agenten möglich sind, kann die Kontrolle über diese Schnittstellen erfolgen. Über die INFORM-Nachrichten erhalten die Kontroll-Knoten Informationen über beginnende oder kürzlich beendete Stromflüsse. Über diese Nachrichten werden zum jetzigen Zeitpunkt die Verbrauchs- und Erzeugungsdaten der untergeordneten Gruppe berechnet. Mit einem INFORM weiß ein Kontroll-Knoten, welche Strommenge verhandelt wurde und kennt die kommunizierenden Agenten. Da ein Stromfluss immer einen der Agenten versorgt, kann der Kontroll-Knoten den Verbrauch der Gruppe um die verhandelte Strommenge erhöhen. Ist zudem einer der Agenten ein Erzeuger, wird die Erzeugung ebenfalls erhöht. Analog werden diese Werte bei einem beendeten Stromfluss verringert. Aus den Verbrauchs- und Erzeugungsdaten kann der Kontroll-Knoten die Energiebilanz bilden. Weist diese Bilanz eine Unter- oder Überproduktion auf, soll der Agent in aufbauenden Implementierungen die Kontroll-Knoten seiner Hierarchieebene kontaktieren. Damit sind die Weichen für eine kontrollierende Kontroll-Schicht gestellt.

In diesem Kapitel wurde zu Beginn die allgemeine Vorgehensweise dargelegt, mit dem Ergebnis, dass der Simulator von Grund auf mittels der Programmiersprache *Python* implementiert wird. Anschließend wurde die technische Realisierung der zwei Prototypen beschrieben. Im Zuge der Beschreibung wurde die Abbildung des Agentenverhaltens und der Kommunikation auf Software, die Funktionalitäten der GUI und der Simulations-Klasse sowie die Einbindung einer Kontroll-Schicht tiefergehend betrachtet. Das nachfolgende Kapitel evaluiert nun unter anderem mithilfe der entwickelten Prototypen das theoretische Simulator-Konzept.

5. Evaluierung

In diesem Kapitel wird das in Kapitel 3 auf Seite 23 dargestellte Konzept evaluiert. Mit der Evaluation soll geklärt werden, welche der in Abschnitt 3.2 auf Seite 26 definierten Anforderungen durch das Konzept erfüllt sind. Weiterhin von Interesse sind die Ursachen für möglicherweise nicht erfüllte Anforderungen. Für die Entwicklung des Konzepts wurde die ABMS angewendet. Unterabschnitt 2.3.3 auf Seite 21 hat bereits die Schwierigkeiten bei der Validierung agentenbasierter Modelle beschrieben. Die Evaluation ist dazu wie folgt gegliedert: Der erste Abschnitt des Kapitels befasst sich mit Einschränkungen der Validität und betrachtet damit in erster Linie das theoretische Konzept. Diese Kriterien wurden vorab aufgestellt. Für die weitere Evaluation wird die prototypische Implementierung analysiert. Der zweite Abschnitt nimmt deshalb Bezug auf die beiden realisierten Prototypen. Damit soll in der Evaluation untersucht werden, in welchem Umfang der entwickelte Simulator in dem Projekt *DevOpt* und weiteren Bereichen eingesetzt werden kann. Wie bereits in Unterabschnitt 4.2.1 auf Seite 55 beschrieben, sind im Verlauf der Masterarbeit zwei Prototypen entstanden. Diese unterscheiden sich grob in der Umsetzung der Agenten. So wurden die Agenten in dem ersten Prototypen in einer Schleife ausgeführt, in dem zweiten Prototypen wurden nebenläufige Threads verwendet.

5.1. Einschränkungen der Validität

In diesem Abschnitt wird das theoretische Simulator-Konzept fokussiert und hinsichtlich der Eignung analysiert. Die Evaluation des Konzepts geschieht anhand von zwei Kriterien, die vorab als relevante Punkte identifiziert wurden. Die zwei Kriterien lauten wie folgt: (1) das entwickelte Konzept erfüllt die definierten Anforderungen und (2) das Konzept bildet beliebige Szenarien eines Electric-Grids ab. Nachfolgend werden die Kriterien einzeln abgearbeitet.

Insgesamt wurden 14 Anforderungen an das Simulator-Konzept gestellt. Diese Bedingungen unterteilen sich in zehn funktionale und vier qualitative Anforderungen. Einige der Anforderungen lassen sich nur anhand einer Implementierung überprüfen, weshalb auf diese tiefergehend in dem anschließenden Abschnitt eingegangen wird. Die Anforderungen (F1) bis (F5) stellen Bedingungen an die identifizierten Komponenten eines Electric-Grids. Diese werden auf die fünf Kategorien *Kontroll-Knoten*,

Hausanschluss, *Erzeuger*, *Energiespeicher* und *Verbraucher* abgebildet. Damit sind alle Komponenten eindeutig zuordbar. Die smarten Komponenten sind zudem eindeutig identifizierbare, modulare Einheiten, sodass sie als Agenten aufgefasst werden können. Damit sind sie nach Definition autonom und handeln unabhängig voneinander. Weiterhin spiegeln die modellierten Agenten in ihrem Handeln das Verhalten der smarten Komponenten wider. Als wichtigste Verhaltensweise wurde dabei die gegenseitige Kommunikation angesehen. Mittels der Kommunikation interagieren die Agenten miteinander. Die Interaktionen sind durch Einteilung in Gruppen sowie die Vorgabe eines Kommunikationsprotokolls eingeschränkt. Damit erfüllt das Simulator-Konzept (F1) bis (F5). Die nächsten Bedingungen, (F6) bis (F10), nehmen Bezug auf den Simulationsablauf und werden aus diesem Grund erst in Abschnitt 5.2 auf der nächsten Seite aufgegriffen. Gleiches gilt für die qualitative Anforderung (Q1). Die Bedingungen (Q2) bis (Q4) stellen Forderungen an die Architektur des Konzepts. Wie in Abschnitt 3.5 auf Seite 49 illustriert, wurde eine hierarchische Architektur, in der das Grundverhalten eines Agenten in einer Wurzelklasse enthalten ist, gewählt. Aufgrund dieser Architektur sind die spezifischen Agentenklassen in Modulen abgebildet. Fehler und Anpassungen sind daher gut lokalisierbar. Zudem können neue Klassen in die Hierarchie eingehängt werden. Diese Charakteristika unterstützen eine gute Wart- und Erweiterbarkeit ((Q2) und (Q3)). Analog gilt dies für die Module der Nachrichten- und Verbindungs-Klasse. Ist die Erweiterbarkeit des Ansatzes gegeben, kann durch umfangreichere Veränderungen auch die Domäne gewechselt werden, was (Q4) stützt. Die Erweiterbarkeit und die Domänenunabhängigkeit werden in Kapitel 6 auf Seite 71 erneut aufgegriffen und detailliert beschrieben und diskutiert.

Das vorgestellte agentenbasierte Modell erfasst die smarten Komponenten eines Electric-Grid Ökosystems. Diese smarten Komponenten wurden als Agenten abgebildet und die individuellen Verhaltensweisen, insbesondere die Interaktionen mit anderen Agenten und mit der Umwelt, spezifiziert. Die Spezifikationen der Agenten und deren Verhalten sind dabei angelehnt an die in Abschnitt 2.1 auf Seite 5 beschriebenen Szenarien sowie die drei Abstraktionsebenen eines Electric-Grids. Da die Agenten in separaten Kategorien bzw. Modulen abgebildet sind, ist es möglich, individuelle Instanzen dieser Module zu erstellen. Weiterhin ist es realisierbar, dass diese erstellten Instanzen in unterschiedliche Gruppen eingeteilt und den Gruppen jeweils übergeordnete Kontroll-Knoten zugewiesen werden. Mit dieser Einteilung und Zuordnung ist der Aufbau eines beliebigen Nano-, Micro- und Urban-Grids möglich. Innerhalb der aufgebauten Szenarien können nun die Agenten nach ihren spezifizierten Verhaltensweisen unabhängig voneinander handeln und über das in Abschnitt 3.4 auf Seite 42 vorgestellte Kommunikationsprotokoll miteinander interagieren. Damit sind die Agenten in der Lage, autonom Stromflüsse miteinander zu verhandeln. Auf diese Weise werden beliebige Szenarien eines Electric-Grids durch das Simulator-Konzept erfasst und abgebildet.

Der nachfolgende Abschnitt betrachtet die prototypischen Implementierungen. Diese wurden anhand des vorgestellten Konzepts realisiert und folgen der hierarchischen Architektur. Die Prototypen wirken einerseits unterstützend bei der Evaluation, da es möglich ist, das modellierte Verhalten in einer Simulation zu überprüfen. Andererseits kann ein funktionsfähiger Simulator im weiteren Verlauf des Projekts genutzt werden, um Ansätze für die Kontroll-Schicht zu testen.

5.2. Implementierungen

In diesem Abschnitt werden die beiden realisierten Prototypen tiefergehend untersucht. Die Prototypen wurden beide anhand des in Kapitel 3 auf Seite 23 vorgestellten Konzepts entwickelt und folgen der hierarchischen Architektur aus Abschnitt 3.5 auf Seite 49. Bei der Realisierung wurde insbesondere darauf geachtet, dass das Agentenverhalten präzise abgebildet ist und die Implementierungen keine schwerwiegenden Verletzungen der Anforderungen zur Folge haben. In dem ersten Unterabschnitt wird dazu der Funktionsumfang der Prototypen anhand der Anforderungen (F6) bis (F10) sowie der qualitativen Anforderung (Q1) untersucht. Damit das implementierte Agentenverhalten überhaupt analysiert werden kann, stellt der zweite Unterabschnitt die Testerstellung für automatisierte Tests vor. In den nachfolgenden beiden Unterabschnitten werden die zwei entstandenen Prototypen getrennt voneinander betrachtet. Für jeden Prototypen werden dabei die nicht erfüllten Anforderungen hinsichtlich der Ursache untersucht. Eine Schwierigkeit der Realisierung bestand darin, den Agenten möglichst viel Freiraum zu gewähren, damit diese autonom und unabhängig handeln und interagieren können. Nur so kann mögliches emergentes Verhalten zum Vorschein kommen. Wie in Unterabschnitt 4.2.1 auf Seite 55 bereits erwähnt, baut der zweite Prototyp auf dem Ersten auf und erweitert diesen im Wesentlichen um die Threading-Komponente.

5.2.1. Funktionalitäten der Simulatoren

Dieser Unterabschnitt nimmt noch einmal Bezug auf die funktionalen Anforderungen (F6) bis (F10) sowie die qualitative Anforderung (Q1). Diese Anforderungen beziehen sich nicht direkt auf das theoretische Konzept, sondern auf eine konkrete Implementierung. Die Anforderungen haben gemein, dass sie der Kategorie SOLL zugeordnet sind. Damit definieren die Anforderungen wünschenswerte Charakteristika des Simulators. In (F6) wird gefordert, dass die Simulation ohne Unterbrechungen laufen soll. Mit der Anforderung ist gemeint, dass der Simulator explizit durch den Anwender beendet werden muss. Der Simulator simuliert damit kein vorab definiertes Szenario, sondern baut ein Szenario auf, welches potenziell endlos laufen und

sich während des Betriebs durch Anwenderaktionen ändern kann. In beiden Prototypen handeln die Agenten gemäß ihres Verhaltens und beenden ihre Aktionen erst, wenn der Simulator explizit durch den Anwender beendet wird. Auch Parameteränderungen führen zu keiner Unterbrechung. Die anfallenden Simulationsdaten, also die versendeten Nachrichten, werden dem Anwender über eine GUI visualisiert und bei Auswahl des Debug-Modus in einem erweiterten Umfang in eine separate Datei geschrieben, welche auch nach der Simulation betrachtet und analysiert werden kann. Damit sind (F7) und (F8) jeweils erfüllt. Wie bereits erwähnt, können in beiden Simulatoren die relevanten Parameter, beispielsweise die Erzeugung oder der Aktivitätsstatus, verändert werden. Diese Änderungen erfolgen durch den Anwender, schränken jedoch die Autonomie der Agenten nicht zwingend ein, da diese Art der Anwender, wie in Abschnitt 3.3 auf Seite 34 beschrieben, als Teil der Umwelt angesehen werden kann und die Änderungen somit zu den Interaktionen zwischen den Agenten und ihrer Umwelt zählen. Drückt der Anwender beispielsweise den Ausschaltknopf eines Fernsehers (Verbraucher), reagiert dieser Verbraucher auf den Umwelteinfluss und geht in den inaktiven Status über. Das Hinzufügen und Entfernen wurde aus zeitlichen Gründen in keinem der Prototypen umgesetzt, da diese Änderungen und die jeweilige Reaktion des Systems auch über das An- und Abschalten von Agenten abgebildet werden. Zudem würde ein Hinzufügen und Entfernen voraussichtlich eine Simulationsunterbrechung nach sich ziehen. Eine Lösungsidee für diese Anforderung ist in der Beschreibung von (F10) skizziert. In dem nächsten Unterabschnitt wird auf die implementierten Tests zur Überprüfung des Agentenverhaltens sowie der GUI-Interaktionen eingegangen.

5.2.2. Automatisierte Tests

Um das implementierte Agentenverhalten evaluieren zu können und um Testläufe wiederholt auszuführen, wurden automatisierte Tests entworfen. In diesem Abschnitt wird kurz auf die Testerstellung und die Testabdeckung eingegangen. Bei den Tests handelt es sich um Unit-Tests, in welchen die Module bzw. Klassen einzeln getestet werden, um die Funktionalität der einzelnen Teile zu prüfen [DCPF18]. Die Tests wurden mit der Programmiersprache Python umgesetzt. Getestet wurden alle Agentenklassen, die Nachrichten- und Verbindungs-Klasse sowie die Simulations-Klasse. Es wurde darauf geachtet, jede einzelne Methode mittels Assertions zu überprüfen. Dazu wurde für die Agentenklassen jeweils ein kleines Szenario erstellt, in welchem die Agenten miteinander interagieren konnten. Die dabei ausgetauschten Nachrichten wurden hinsichtlich des Inhalts und der Kommunikationsabsicht geprüft. Durch diese Prüfung können Rückschlüsse auf das Agentenverhalten gezogen werden, da die Agenten als wesentliches Verhalten die Möglichkeit zur Kommunikation haben. Die Nachrichten- und Verbindungs-Klasse enthalten lediglich Getter- und Setter-Methoden, was die zugehörigen Tests relativ simpel gestaltet. Für die Simulations-

Klasse musste überprüft werden, ob das Szenario korrekt erstellt wurde und alle beabsichtigten Hierarchien zwischen den Agenten stimmen. Dazu wurden Agenten kreiert, die nach der Simulationserstellung hinsichtlich ihrer Kommunikationspartner, dem zugeordneten Kontroll-Knoten und den untergeordneten Agenten überprüft wurden. Die entworfene Testsuite kann für beide Protoypen gleichermaßen angewandt werden, weil das Verhalten der Agenten, nicht aber die konkrete Implementierung getestet wird. Da nahezu jede Klasse und Methode, mit Ausnahme der GUI, getestet wurde, ist die Testabdeckung sehr hoch. Die GUI-bezogenen Klassen sowie die Methoden zur Interaktion zwischen Anwender und Simulator wurden an der laufenden Software getestet. Nachfolgend wird die erste prototypische Implementierung näher betrachtet.

5.2.3. Erste prototypische Implementierung

In der ersten prototypischen Implementierung werden die Agenten nach Erstellung des Electric-Grids in einer Liste gespeichert. Diese Liste wird nach dem Start der Simulation iterativ in einer Schleife durchlaufen. Die Schleife wird erst durch einen Eingriff des Anwenders unterbrochen. Somit läuft die Simulation ohne Unterbrechungen ab. Die Schleife erzwingt jedoch eine Art Ausführungsreihenfolge. Diese Reihenfolge und die damit verbundene rundenbasierte Abarbeitung der Agentenhandlungen widersprechen der Autonomie eines Agenten und verletzen die Anforderung (F4). Damit ist eine Anforderung aus der Kategorie MUSS nicht erfüllt. Als Folge dessen kann es passieren, dass der Simualator nicht das geforderte Agentenverhalten widerspiegelt. Die Autonomie ist allerdings keine Voraussetzung zur Entstehung von emergentem Verhalten. Nach Weyer und Roos [WR17] reichen bereits simple Verhaltensweisen der Systemkomponenten, in diesem Fall der Agenten, aus, damit ein Gesamtsystem emergentes Verhalten ausbildet. Als Ursache von Emergenz wurden in Unterabschnitt 2.2.2 auf Seite 14 die Interaktionen der Agenten miteinander und mit ihrer Umwelt genannt. Diese Interaktionen finden auch in dem ersten Prototypen statt, da die Agenten die Möglichkeit zur Kommunikation besitzen. In dem Prototypen besteht außerdem die Möglichkeit, die Simulationsparameter während der laufenden Simulation anzupassen und die Reaktion des Systems zu beobachten. Weiterhin bietet die gewählte Architektur die Möglichkeit, dass die Regeln der Kontroll-Schichten spezifiziert werden können. Über Schnittstellen können die Kontroll-Knoten auf die Agenten einwirken. Der erste Prototyp erfüllt zwar nicht alle Anforderungen, er kann jedoch trotzdem verwendet werden, um Ansätze für die Kontroll-Schicht zu testen. Weiterhin bietet er die Möglichkeit, das Agentenverhalten, mit Ausnahme der Autonomie, zu visualisieren und damit das agentenbasierte Modell zu evaluieren. Nachfolgend wird die Implementierung des zweiten Prototypen hinsichtlich der Autonomie evaluiert.

5.2.4. Zweite prototypische Implementierung

Bei der Realisierung des zweiten Prototypen bestand das Ziel darin, mittels des Simulators die Realität näher abzubilden. Dazu wurde in erste Linie versucht, die Autonomie der Agenten zu erreichen und damit die Anforderung (F4) zu erfüllen. Mit Autonomie ist dabei gemeint, dass die Agenten autonom handeln können und nicht warten müssen, bis sie erneut an der Reihe sind. In dem zweiten Prototypen verbleibt, dass die Agenten durch den Anwender in den aktiven bzw. inaktiven Zustand versetzt werden und beispielsweise der Verbrauch verändert wird. Nach Unterabschnitt 3.3.3 auf Seite 38 zählen diese Anwenderaktion jedoch zu den Interaktionen mit der Umwelt und stellen damit keinen Eingriff in die Autonomie dar, obwohl die Agenten durch diese Interaktionen und die damit verbundenen Sperren ausgebremst werden. Um die Autonomie der Agenten zu erreichen, läuft der zweite Prototyp mit mehreren unabhängigen Threads. Statt die Agenten in einer Liste zu speichern und die Handlungen iterativ und in einer vorgegebenen Reihenfolge abzuarbeiten, läuft jeder Agent in einem separaten Thread. Durch diese separaten Threads handeln die Agenten unabhängig voneinander und sind an keine Ausführungsreihenfolge gebunden. Die Agenten interagieren autonom. Damit ist in dem zweiten Prototypen, in Bezug auf das reine Agentenverhalten, die Anforderung (F4) im Gegensatz zu dem Ersten erfüllt. Die Threads und die damit verbundene parallele Ausführung der Agenten spiegelt zudem die realen, modularen IoT-Devices wider. Da keine weiteren nennenswerten Veränderungen im Vergleich zu dem ersten Prototypen erfolgt sind, sind die übrigen Anforderungen ebenfalls für den zweiten Prototypen erfüllt. Insgesamt lässt sich festhalten, dass auch dieser Prototyp das Agentenverhalten abbildet und dabei realen IoT-Devices noch ein Stück näher kommt. Die zweite prototypische Implementierung kann damit ebenfalls als Testumgebung für die zu spezifizierende Kontroll-Schicht eingesetzt werden.

In den Testläufen des zweiten prototypischen Simulators sind bereits neue, unbeabsichtigte Verhaltensweisen aufgetreten, wenn in einem Szenario mehrere Verbraucher und Erzeuger der gleichen Gruppe angehörten und miteinander kommunizieren konnten. Die neuen Verhaltensweisen zeichneten sich dadurch ab, dass die Agenten zum Teil Schwierigkeiten hatten, einen virtuellen Stromfluss aufrechtzuerhalten. Als Ursache wurde dabei die zeitgleiche Kommunikation mit mehreren Agenten verzeichnet. Ein Verbraucher hat beispielsweise als Folge dessen eine Verbindung zu mehreren Erzeugern aufgebaut. Nach den spezifizierten Regeln wird ein Verbraucher jedoch nur von maximal einem Erzeuger versorgt. Aus diesem Grund wurden, um keinen der Erzeuger bevorzugt zu behandeln, alle virtuellen Stromflüsse beendet. Da der Verbraucher nun nicht mehr versorgt ist, beginnt das Protokoll und damit die Überversorgung von Neuem. Es wurde bewusst entschieden, dieses Verhalten nicht durch Ignorieren von eingehenden Nachrichten während einer Verhandlung oder die Bevorzugung eines Erzeugers zu unterdrücken. Der Grund für diese Ent-

scheidung war, dass es sich bei diesem unbeabsichtigten Verhalten bereits um eine Form der Emergenz handeln kann. Für einen kontrollierten Ablauf und eine erfolgreiche Stromverhandlung wäre somit die Kontroll-Schicht zuständig.

Eine Frage, die bei dem zweiten Prototypen noch offen bleibt, ist die Skalierbarkeit des gewählten Ansatzes. Da in dieser Implementierung für jeden einzelnen Agenten ein eigener Thread erstellt wird, steigt die Zahl der Threads linear an. Bei großen Szenarien werden durch diesen Anstieg eher die Grenzen der Hardware erreicht, als dies für den ersten Prototypen der Fall ist. Die Analyse und Verbesserung der Performanz der Implementierungen übersteigt jedoch den Rahmen dieser Masterarbeit. Da in beiden Prototypen das Agentenverhalten abgebildet wird, hat jede der Implementierungen ihre Berechtigung.

In diesem Kapitel wurde das Simulator-Konzept evaluiert. Dazu wurde zu Beginn des Kapitels anhand des theoretischen Konzepts überprüft, welche der 14 Anforderungen erfüllt sind. Da einige der Anforderungen nur mittels konkreter Implementierungen des Konzepts geprüft werden können, hat der zweite Abschnitt die realisierten Prototypen fokussiert. Insgesamt lässt sich aus der Evaluation ableiten, dass das Simulator-Konzept und die prototypischen Implementierungen die Agenten, die Umwelt sowie die jeweiligen Interaktionen eines Electric-Grids mit wenigen Einschränkungen abbilden. Damit eignet sich das Simulator-Konzept für den beabsichtigten Zweck. Das nun anschließende Kapitel greift noch einmal die relevantesten Kritikpunkte auf und diskutiert diese ausführlich.

6. Diskussion

In dem bisherigen Teil der Arbeit wurde das Thema vorgestellt und motiviert, Grundlagen zum Verständnis gebildet und das Konzept für den Szenarien-Simulator sowie die Implementierung vorgestellt und evaluiert. Dieses Kapitel diskutiert Kritikpunkte an dem vorgestellten Szenarien-Simulator und den gestellten Anforderungen. Dabei fokussiert das Kapitel auf vier Diskussionspunkte. Als erstes werden die Einschränkungen des Konzepts beleuchtet. Der zweite Abschnitt stellt alternative Ansätze vor, welche sich von den in Abschnitt 1.1 auf Seite 2 unterscheiden. Nachfolgend wird auf die geforderte Erweiterbarkeit eingegangen und zum Schluss der Diskussion die Domänenunabhängigkeit des Konzepts hervorgehoben.

6.1. Einschränkungen des Simulators

Die Anforderungen an das Simulator-Konzept basieren auf den zuvor beschriebenen und gründlich analysierten Szenarien. In dem Simulator-Konzept wurden somit der grundlegende Aufbau der drei Grids sowie die Handlungen und Interaktionen der IoT-Devices und Kontroll-Knoten während des Betriebs berücksichtigt. Wie in Kapitel 5 auf Seite 63 herausgestellt, ist das vorgestellte Konzept in der Lage, die Agenten eines Electric-Grid Ökosystems und ihre Umwelt in der nötigen Granularität zu erfassen und zu simulieren. Damit wurde die generelle Anwendbarkeit überprüft. An dieser Stelle sollte jedoch festgehalten werden, dass das Simulator-Konzept ein Modell eines realen Systems darstellt und somit vereinfachte Annahmen und Abstraktionen beinhaltet.

Die wohl relevanteste Vereinfachung ist die Annahme einer heterogenen IoT-Landschaft. So sind unter anderem die Hardwarespezifikationen der realen IoT-Devices nicht in die Entwicklung eingeflossen. Damit kann der Simulator keine gerätetypischen Latenzzeiten darstellen. Der Grund für diese Vereinfachung ist die technische Umsetzung. Um in diesem Punkt näher an die Realität zu gelangen, sei ein physischer Demonstrator genannt, welcher die IoT-Devices eines Electric-Grids auf Hardwarekomponenten abbildet. Der nachfolgende Abschnitt erläutert genauer, inwiefern ein Demonstrator anstelle des Simulators verwendet werden kann. Die Einflüsse der Kontroll-Schicht finden jedoch rein softwareseitig statt, sodass diese technische Abstraktion keine Einschränkung für die Eignung des Simulators darstellt.

Eine weitere vereinfachte Annahme ist die grobe Einteilung der IoT-Devices in die fünf Kategorien Verbraucher, Erzeuger, Speicher, Hausanschluss und Kontroll-Knoten. Durch die Einteilung wird beispielsweise ein riesiges Kraftwerk mit einem heimischen Erzeuger vom Verhalten her gleichgesetzt. Diese Annahme ist eine starke Abstraktion der Realität. Für das Testen von Ansätzen für die Kontroll-Schicht stellt dies jedoch keine Einschränkung dar. Die Kontroll-Schicht soll die Verwaltung der Stromflüsse überwachen und ändern können, um Optimierungen innerhalb des Systems vorzunehmen. Für diese Aufgabe ist es ausreichend zu wissen, wer Strom erzeugt und in welchen Mengen. Wie die Erzeuger ihren produzierten Strom anbieten, spielt dabei keine Rolle. Analog gilt dies für Verbraucher, Speicher und Hausanschluss. Inwiefern eine weitere Aufsplittung der Kategorie Kontroll-Knoten nötig ist, wird sich im Verlauf des Projekts zeigen.

6.2. Alternatives Feedbacksystem

In Abschnitt 2.2 auf Seite 12 wurde bereits herausgestellt, dass die Simulation eines nachgebildeten Systems eine geeignete Methode für die Erkennung von Emergenz und zur Abschätzung der Reaktion auf Änderungen darstellt. Abschnitt 2.3 auf Seite 16 hat weiterhin herausgestellt, dass die ABMS ein gutes Maß an Komplexität besitzt, damit gute Modelle entwickelt werden können. Für die softwarebasierte Simulation sprechen zudem die unzähligen existierenden Ansätze (Abschnitt 1.1 auf Seite 2 und Abschnitt 4.1 auf Seite 53)

Eine Alternative zu der softwarebasierten Simulation stellt ein physischer Demonstrator dar. Wie bereits erwähnt, ist ein physischer Demonstrator eine hardwarebasierte Nachbildung des realen Systems und seiner Umgebung. Für die Realisierung können Microcontroller als Hardware verwendet werden. Im Fall eines Electric-Grids könnte eine Hardwarekomponente beispielsweise ein IoT-Device und dessen Verhalten simulieren. Physische Demonstratoren eignen sich prinzipiell für die Imitierung des Systemverhaltens. Die Hardware ermöglicht es, die interagierenden Agenten darzustellen und das Verhalten zu beobachten. Zudem können Updates und weitere Änderungen in den Demonstrator eingespielt und die Reaktion auf die neuen Bedingungen untersucht werden. Die Nachbildung mittels Hardware hat den Vorteil, dass der beschriebene Ansatz anschaulich dargestellt ist. Physische Demonstratoren haben weiterhin, wie in dem vorstehenden Abschnitt angesprochen, die Eigenschaft, dass hard- und netzwerkspezifische Latenzen und Überlastungen zum Teil abgebildet werden. Die Verwendung von phyischen Demonstratoren hat jedoch eine aufwendige Testphase zur Folge, da diese Demonstratoren in der Regel nicht direkt am Arbeitsplatz aufgebaut sind. Weiterhin ist das dargestellte Szenario fest, sodass ein physischer Demonstrator wenig flexibel ist und neue Szenarien eine Umbauphase mit

sich bringen. Des Weiteren ist die notwendige Hardware für den Aufbau teuer. Diese aufwendige und teure Testphase schränkt den Arbeitsablauf der Entwicklerteams ein und kann somit die Bereitstellung neuer Updates verzögern.

Ein physischer Demonstrator bringt einige Vorteile mit sich, resultiert aber in einem langen Feedbacksystem. Ziel dieser Masterarbeit war jedoch die Entwicklung eines kurzen Feedbacksystems, was zum Ausschluss der Demonstratoren führte.

6.3. Erweiterbarkeit des Simulator-Konzepts

Den Schluss der Diskussion bildet die Auseinandersetzung mit der Erweiterbarkeit des Ansatzes. Erweiterbarkeit ist eine nach der Anforderungsdefinition unter dem Punkt (Q3) gewünschte Eigenschaft. Die Erweiterbarkeit des Konzepts bezieht sich dabei einerseits auf die Integration neuer (Kommunikations-)Protokolle und andererseits auf die Einbindung neuer IoT-Devices und Device-Kategorien. Anhand zweier Beispiele werden diese beiden Erweiterungen nun exemplarisch vorgestellt.

Einbindung neuer Protokolle

Im Folgenden wird die Einbindung eines neuen Protokolls anhand der ACL vorgestellt. Das ACL-Protokoll ist, wie in Abschnitt 4.1 auf Seite 54 erwähnt, in dem Framework PADE implementiert. Um Zugriff auf diese Implementierung zu erhalten, können die benötigten Klassen einfach in die Klassenhierarchie des Simulator-Konzepts eingehängt werden, sodass die eigens definierte Nachrichten-Klasse von der zuständigen PADE-Klasse erbt. Die Vererbung an dieser Stelle ist sinnvoll, da so unnötige Änderungen des Variablentyps innerhalb des Simulator-Codes ausbleiben. Im Anschluss müssen gegebenfalls Änderungen in den Agenten-Klassen erfolgen, damit die nötigen Parameter der ACL-Nachricht spezifiziert werden. Diese Vorgehensweise ist nicht nur auf die ACL beschränkt, sondern prinzipiell allgemeingültig. Möglich macht dies das hierarchische Simulator-Konzept.

Integration neuer Devices und Kategorien

An dieser Stelle wird die Einbindung eines neuen IoT-Devices exemplarisch dargestellt. Unter neue IoT-Devices fallen Spezialisierungen der fünf bereits existierenden Kategorien. Die Spezialisierungen bieten den Vorteil, dass das Grundverhalten bereits vorhanden ist und lediglich ergänzt werden muss. Abbildung 6.1 auf der nächsten Seite demonstriert das Hinzufügen eines mobilen Verbrauchers. Ein mobiler Verbraucher ist beispielsweise ein Smartphone oder ein Elektrofahrzeug. Das

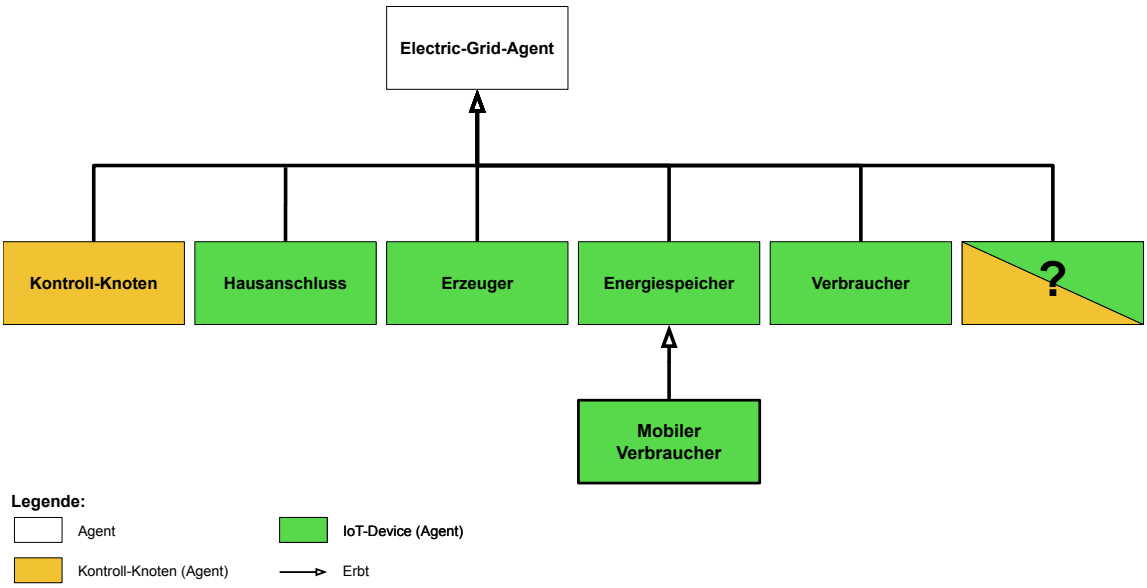


Abbildung 6.1.: Integration eines neuen IoT-Device. Die Abbildung zeigt exemplarisch die Integration eines neuen IoT-Device in das Konzept. Dies wird zum einen visualisiert durch das Hinzufügen des mobilen Verbrauchers. Der mobile Verbraucher erbt dabei die Eigenschaften und Verhaltensweisen des Electric-Grid-Agenten und erweitert diese unter anderem um die Fähigkeiten zu laden und die Gruppe zu wechseln. Zwei Beispiele für konkrete mobile Verbraucher sind ein Elektrofahrzeug und ein Smartphone. Zum anderen wird die Integration durch Hinzufügen einer neuen Kategorie dargestellt. Die neue Kategorie erbt das Grundverhalten des Electric-Grid-Agenten und kann dadurch sowohl der IoT-Schicht (grün) als auch der Kontroll-Schicht (orange) zugehörig sein.

Grundverhalten des mobilen Verbrauchers ähnelt dem Verhalten eines regulären Verbrauchers: Der mobile Verbraucher kann kommunizieren. Wenn der Verbraucher Strom benötigt, kontaktiert er die IoT-Devices in seiner Nachbarschaft. Nachdem der Verbraucher keinen Strom mehr benötigt, unterbricht er die Versorgung. Den mobilen Verbraucher unterscheidet dabei, dass dieser über einen Akku betrieben wird. Somit benötigt der mobile Verbraucher Strom, wenn der eigene Akku entladen ist bzw. der Ladestand einen kritischen Wert unterschreitet. Ein mobiler Verbraucher benötigt somit weiteres Verhalten, welches ähnlich zu dem vorhanden Speicher ist. Von dieser Kategorie kann jedoch nicht geerbt werden, da der Verbraucher im Gegensatz zum Speicher keine weiteren Devices laden kann. Weiteres Verhalten könnte die Bewegung des Agenten, z.B. eines Autos, sein. Ein interessanter Aspekt des mobilen Verbrauchers ist, dass dieser sich während der laufenden Simulation in unterschiedlichen Gruppen aufhalten kann. Denn üblicherweise trägt man sein Smartphone auch außerhalb der eigenen Wohnung bei sich oder durchquert während der Autofahrt

mehrere Stadtteile oder gar Städte. Als Folge dieser Bewegungen ändern sich sowohl die Kommunikationspartner als auch der zuständige Kontroll-Knoten. Auch unterwegs sollte die Kommunikation zwischen dem mobilen und den jeweils lokalen IoT-Devices funktionieren, damit ein Laden des mobilen Verbrauchers möglich ist. Beispielsweise kann durch frühes Ankündigen des Ladestandes und der Verkündung der nachfolgenden Aufenthaltsorte der Ladevorgang geplant werden. Ein Elektrofahrzeug kann damit auf einer bekannten Route gezielt in den Bezirken geladen werden, in denen momentan viel (Öko)-Strom verfügbar ist.

Eine weitere Möglichkeit der Erweiterung ist die Integration einer neuen Agenten-Kategorie. Die neue Kategorie erbt das Grundverhalten des Electric-Grid-Agenten und hat damit im Wesentlichen die Fähigkeit zur Kommunikation mit Agenten der eigenen Nachbarschaft. Die neu hinzugefügte Kategorie kann sowohl der IoT-Schicht (grün) als auch der Kontroll-Schicht (orange) zugehörig sein, da der Wurzelknoten diesbezüglich keine Einschränkungen vornimmt. Die dargestellte Vorgehensweise ist auf beliebige Integrationen von IoT-Devices anzuwenden. Damit ist bewiesen, dass das Konzept erweiterbar ist und die Anforderung (Q3) erfüllt.

6.4. Domänenunabhängigkeit

Ein weiteres Diskussionsthema ist die Domänenunabhängigkeit. Diese wird in Abschnitt 3.2 auf Seite 26 unter der Anforderung (Q4) definiert und für das Konzept gefordert. Vorweg sei angemerkt, dass vollständige Domänenunabhängigkeit nicht erreicht werden kann, da die beobachteten Agenten und ihr zugehöriges Verhalten in Hinblick auf die Verwendung in einem Electric-Grid zu spezifisch sind und somit auch nicht direkt auf verwandte Domänen, wie die smarte Wasserversorgung, angewendet werden können. Interessant für die Domänenunabhängigkeit ist jedoch, inwieweit das Konzept erweiterbar ist und die theoretischen Grundlagen und die Herangehensweise übertragbar sind.

In dem vorgestellten Konzept sind die Kontroll- und die IoT-Schicht in verschiedenen Klassen implementiert. Der unmittelbar vorangegangene Abschnitt hat anhand des mobilen Verbrauchers und des ACL-Protokolls die Erweiterbarkeit des vorgestellten Ansatzes präsentiert. Durch die Erweiterbarkeit können einzelne Komponenten ersetzt werden. Weiterhin liegt die Kontroll-Schicht innerhalb der Simulation virtuell über der IoT-Schicht und greift über definierte Schnittstellen in die Szenarien ein. Somit ist sowohl eine Austauschbarkeit der IoT-Szenarien gegeben als auch die übersichtliche Anpassbarkeit und Austauschbarkeit der Agenten-Klassen. Damit ist Unabhängigkeit bezüglich des Electric-Grid Szenarios gegeben.

Die Erweiterbarkeit ermöglicht außerdem, dass die IoT-Schicht gegen eine IoT-Landschaft einer verwandten Domäne ausgetauscht wird. Aufgrund der definierten Schnittstellen ist die verbleibende Kontroll-Schicht prinzipiell weiterhin in der Lage, in das Szenario einzugreifen. Da sich die Domäne und damit auch die internen Regeln unterscheiden, wird dieser Eingriff jedoch möglicherweise keine (positiven) Effekte auf das System haben. Um dies zu erreichen, müssen die Regeln der Kontroll-Schicht angepasst werden, was durch die Erweiterbarkeit möglich ist. Gegebenenfalls müssen noch Schnittstellen angepasst oder neu definiert werden.

Die Theorie lässt sich zusammenfassen in (1) die ABMS, mit welcher Emergenz und die Reaktionen des Systems auf Änderungen sichtbar werden sollen, (2) die Hierarchiebildung bei der Kommunikation der IoT-Devices und (3) die Verwendung einer übergeordneten Kontroll-Schicht. Macal und North [MN08] empfehlen die ABMS, wenn innerhalb des (selbstorganisierenden) Systems Agenten identifizierbar sind, welche ein individuelles Verhalten haben und untereinander und mit ihrer Umwelt interagieren. Weiterhin bietet sich die Methode an, wenn während des Betriebs Emergenz zu erwarten ist. Damit ist der Punkt (1) auf verwandte Domänen, wie die smarte Wasserversorgung, anwendbar. Bei der Wasserversorgung können unter anderem die Drucksensoren als Agenten repräsentiert werden. Sofern Agenten identifizierbar sind, kann innerhalb der Menge eine Hierarchiebildung stattfinden. Damit ist Punkt (2) übertragbar. Innerhalb einer Hierarchie können einzelne Schichten als Kontroll-Schicht implementiert werden. Diese Eigenschaft erfüllt Punkt (3).

Der Simulator ist in der Lage, diverse Szenarien eines Electric-Grids zu simulieren. Da der Eingriff der Kontroll-Schicht über definierte Schnittstellen erfolgt, ist es möglich, die IoT- oder Kontroll-Schicht auszutauschen. Mit der Austauschbarkeit der Schichten sind nicht nur Szenarien eines Electric-Grids, sondern auch weiterer Domänen simulierbar. Weiterhin sind die Grundlagen und die Herangehensweise bei der Entwicklung auf andere Bereiche übertragbar. Die Domänenunabhängigkeit des vorgestellten Simulator-Konzepts ist somit erfüllt.

In der Diskussion wurde zunächst auf die Einschränkungen dieses softwarebasierten Simulators eingegangen. Die Gründe für diese Einschränkungen lassen sich grob technischer oder modellierungsbedingter Natur zuordnen. Die diskutierten Einschränkungen haben jedoch keinen Einfluss auf den beabsichtigten Einsatzzweck. Als mögliche Alternative wurde im Folgenden ein physischer Demonstrator beschrieben. Dieser kann die technischen Defizite zum Teil beheben, lässt sich jedoch nur schwer erweitern. Erweiterbarkeit resultiert in einem flexibleren Ansatz, weshalb im nächsten Abschnitt die Erweiterbarkeit anhand zweier Beispiele präsentiert wurde. Aufbauend auf die Erweiterbarkeit wurde als letzter Punkt die Domänenunabhängigkeit hervorgehobenen. Das nachfolgende, letzte Kapitel fasst diese Masterarbeit kurz zusammen und stellt mögliche zukünftige Anknüpfungen vor.

7. Zusammenfassung und Ausblick

Die vorgestellte Masterarbeit hat die Erstellung eines Simulator-Konzepts für ein Electric-Grid IoT-System thematisiert. In der Einleitung wurde die Notwendigkeit eines Simulators als kurzes Feedbacksystem für komplexe, adaptive Systeme deutlich. Diese Notwendigkeit wurde gestützt durch die Vorstellung diverser bereits existierender Simulatoren. Das zweite Kapitel hat nach der Vorstellung der Electric-Grid Ebenen die fundamentalen Konzepte Selbstorganisation sowie Emergenz erklärt und mit der ABMS eine Methode präsentiert, die mögliche Emergenzen aufdecken kann. Anknüpfend an diese Grundlagen hat das dritte Kapitel die Entwicklung eines Simulator-Konzepts zur Modelierung und Simulierung eines Electric-Grids dargestellt. Dazu wurden zu Beginn Szenarien eines Electric-Grids analysiert. Aus dieser Analyse sowie den Grundlagen wurden nachfolgend Anforderungen an den späteren Simulator gestellt. Den Schwerpunkt des Kapitels bildete die Erstellung des agentenbasierten Modells unter Anwendung der ABMS. Anschließend wurde das entwickelte Kommunikationsprotokoll und die Gesamtarchitektur des Konzepts diskutiert. In Kapitel vier wurde eine mögliche Realisierung vorgestellt. Im Zuge dieser Vorstellung wurden Fragen wie die Wahl der Programmiersprache und die technische Umsetzung thematisiert. Die zwei bei der Realisierung entstandenen Prototypen wurden in der Evaluation in Kapitel fünf aufgegriffen und bezüglich ihrer Validität und der Erfüllung der zuvor definierten Anforderungen untersucht. In dem vorletzten, sechsten Kapitel wurden relevante Diskussionspunkte des Konzepts aufgegriffen und tiefergehend betrachtet. Die Diskussion hat insbesondere die Erweiterbarkeit und die daraus resultierende Domänenunabhängigkeit des vorgestellten Konzepts herausgestellt.

Die Forschungsfrage der Masterarbeit lautete, inwiefern ein IoT-System modelliert und in einem softwarebasierten Simulator abgebildet werden kann, sodass potenzielles emergentes Verhalten während der Simulation zum Vorschein kommt. Dabei war ebenfalls von Interesse, welche Einschränkungen bei einem rein softwarebasierten Ansatz eingegangen werden und welche Ursachen diesen Einschränkungen zugrunde liegen. Die Forschungsfrage kann wie folgt beantwortet werden: Ein IoT-System und damit auch ein Electric-Grid kann softwarebasiert modelliert und simuliert werden. Relevant für die Eignung ist zum einen die korrekte Abbildung der Komponenten eines Electric-Grids und zum anderen der Funktionsumfang des Simulators (z.B. Visualisierung der Interaktionen). Für die korrekte Umsetzung greift das vorgestellte Konzept auf die Verwendung von miteinander interagierenden Agenten zurück. Die spezifizierten Software-Agenten können alle Kommunikationsabläufe abbilden und

bieten durch die Modularisierung und die hierarchische Architektur eine sehr gute Anpassbarkeit. Die Evaluation hat aufgezeigt, dass in dem Gesamtsystem bereits bei Verwendung weniger Agenten emergentes Verhalten auftritt. Diese wird über die zugehörige GUI sichtbar. Der Simulator hat jedoch auch, wie in der vorangegangenen Diskussion beschrieben, Einschränkungen. Diese sind zum einen zurückzuführen auf technische Limitierungen, z.B. die Abbildung spezifischen Hardwareverhaltens und Latenzzeiten. Zum anderen liegt die Ursache in der Realisierung. Die vorhandenen Einschränkungen des softwarebasierten Ansatzes sind jedoch nicht relevant für die Funktionsweise und Eignung des Simulators.

Einen möglichen Anknüpfungspunkt stellt die Spezifikation der Kontroll-Schicht dar. Damit könnte der Forschungsfrage nachgegangen werden, wann und wie die Kontroll-Schicht eingreifen soll und welche Regeln zur Erreichung dieser Kontrolle benötigt werden. Die Regeln sind von großer Relevanz, da sie dabei helfen, emergentes Verhalten zu lenken und eine globale Optimierung der Ressourcennutzung zu erreichen. Weiterhin nicht zu vernachlässigen ist die immense Rechenleistung einiger IoT-Devices. Die Rechenleistung moderner Komponenten kann von großer Bedeutung in Hinblick auf intelligente, lernende Agenten sein. Die Regeln innerhalb des Systems müssten damit nicht vorab gesetzt werden, sondern können sich während des Betriebs bilden. Anknüpfende Arbeiten könnten zudem die Auslagerung von (Kontroll)-Aufgaben auf leistungsstarke IoT-Devices behandeln. Weiterhin ist die Entwicklung eines Konzepts für die Registrierung mobiler Agenten bei ihren Kontroll-Knoten denkbar. Diese mobilen Verbraucher wurden in der Diskussion bereits vorgestellt und es wurden Möglichkeiten zur Realisierung und Einbindung illustriert. In dem in Unterabschnitt 3.4.1 auf Seite 44 vorgestellten Prozess fehlen jedoch Mechanismen, damit ein Agent mitbekommt, wann er seine Gruppe wechselt. Dazu gehört auch, dass gegebenenfalls der frühere Kontroll-Knoten über den Wechsel informiert wird. Agentenbewegungen im Allgemeinen könnten in der Zukunft weitere Bedeutung gewinnen, beispielsweise durch vermehrten Einsatz von Elektrofahrzeugen oder der Einbindung von Smartphones.

Des Weiteren bieten sich auf der Implementierungsebene vielfältige Möglichkeiten zur Erweiterung und Optimierung der Simulatoren. In Unterabschnitt 5.2.4 auf Seite 68 wurde bereits die offene Frage der Skalierbarkeit sowie der Performanz der entwickelten Prototypen angesprochen. Für eine Beantwortung müssten die Implementierungen tiefergehend hinsichtlich der Ressourcennutzung analysiert werden. In Kombination mit einer ausführlichen Recherche sind gegebenenfalls Optimierungen möglich. In einer aufbauenden Version des Simulators ist außerdem denkbar, dass vorab definierte Lastgänge in das System geladen werden können. Diese Lastgänge beinhalten für jeden Zeitpunkt eine Arbeitskonfiguration für die Agenten der Simulation. Damit ist eine Simulation mit den gleichen Parametern wiederholt ausführbar. Zwischen den Simulationsläufen ist es beispielsweise möglich, die Regeln der Kontroll-Schicht zu verändern und das resultierende Systemverhalten zu beob-

achten. Dies bietet den Vorteil, dass bei gleichbleibenden Regeln überprüft werden kann, ob eventuelle Emergenzen reproduzierbar sind. Für diese Anpassung muss jedoch vorerst der Aspekt von Zeit und Zeitpunkten spezifiziert werden.

Zusammenfassend lässt sich festhalten, dass das erstellte Simulator-Konzept in der Lage ist, eine IoT-Landschaft adäquat zu erfassen und abzubilden. Damit bietet es eine sehr gute Basis für kommende Testläufe der Kontroll-Schicht. Der Simulator erfasst mittels eines agentenbasierten Modells das Verhalten der IoT-Devices und bietet die Möglichkeit zur Einbindung einer Kontroll-Schicht. In dieser Kontroll-Schicht können durch die sehr gute Erweiterbarkeit im Nachhinein neue Regeln spezifiziert werden. Die Erweiterbarkeit wirkt sich außerdem positiv auf die Domänenunabhängigkeit des vorgestellten Ansatzes aus. Damit sind nicht nur Electric-Grids, sondern beliebige IoT-Ökosysteme modellierbar und simulierbar. Die Masterarbeit grenzt sich insbesondere in dem beabsichtigten Verwendungszweck als Testumgebung für mehrstufige Kontroll-Schichten von bisherigen Ansätzen, existierenden Simulatoren und verfügbaren Plattformen ab. Zudem kann der geschriebene Programmcode sowohl für einen rein softwarebasierten Simulator, als auch mit geringen Anpassungen für einen physischen Demonstrator verwendet werden und sich damit zukünftig als sinnvoll erweisen.

A. Anhang

A.1. DevOpt-Projektsteckbrief

Bundesministerium
für Bildung
und Forschung

BMBF – Fördermaßnahme
Konstruktionsprinzipien und Laufzeitmethodik

Projekt: DevOpt – DevOps für Selbst-Optimierende Emergente Systeme

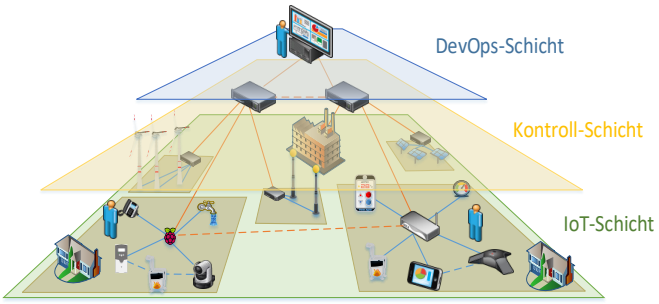
Neuen Ideen für offene, emergente und dynamisch veränderliche IT-Systeme

Softwaresysteme z. B. im Internet der Dinge (IoT) werden zunehmend komplexer und dezentraler. Um damit verbundene Herausforderungen zu meistern, muss die Entwicklung resilienter Systeme vorangetrieben werden. Darunter sind solche Systeme zu verstehen, die durch einen adäquaten Umgang mit Störungen, Unsicherheiten und auch widrigen Bedingungen, wie z. B. eingeschränkten Ressourcen (CPS, IoT) oder Fehlbedienung, einen Betrieb aufrechterhalten können („Fail Operational“). Hierzu sind innovative Architekturen und Designprinzipien sowie Betriebs- bzw. Laufzeitumgebungen erforderlich, die im Sinne der Selbstorganisation u. a. auf Prinzipien aus dem Organic Computing und der Bionik zurückgreifen, um Strukturen von biologischen Organismen auf die Informationstechnologie zu übertragen. Mit zwei Bekanntmachungen zu Resilienz und Ausfallsicherheit sowie zu Konstruktionsprinzipien und Laufzeitmethodik möchte das BMBF die Entwicklung von neuen Methoden und Werkzeugen für offene, emergente und dynamisch veränderliche IT-Systeme unterstützen. Mit der Förderung dieser Forschungsvorhaben schafft das BMBF die Grundlage für weitere Innovationen mit hohem gesamtwirtschaftlichem Potenzial und trägt damit zur Stärkung des Wirtschafts- und Wissenschaftsstandortes Deutschland bei.

DevOps für Selbst-Optimierende Emergente Systeme (*DevOpt*)

Durch die kontinuierlich wachsende Komplexität von softwareintensiven Systemen stoßen klassische Methoden und Verfahren der Informatik zunehmend an ihre Grenzen. Heutige softwareintensive Systeme sind zunehmend als Teil eines größeren *IT-Ökosystems* zu betrachten, das kontinuierlich verändert wird, da Betrieb und Entwicklung zunehmend miteinander verschränkt werden (*DevOps*). Daher bestehen diese IT-Ökosysteme im Gegensatz zu traditionellen, hierarchischen Systemen nicht aus Subsystemen, deren Interaktionen in der Regel global plan-, entwerf- und steuerbar sind. Vielmehr sind sie komplexe, adaptive Systemverbünde von interagierenden autonomen Einzelsystemen (‚*smart ecosystems*'). Ein wichtiges Merkmal dieser Systeme ist die Entstehung emergenten Verhaltens durch das Zusammenspiel der Elemente, d.h. Verhalten, das nicht detailliert vorgegeben ist, sondern im System selbst entsteht. Im Verlauf des Betriebs wird das System zudem kontinuierlich verändert, so dass sich dieses jeweils möglichst autonom anpasst.

Im Projekt DevOpt wird ein Ansatz für die Entwicklung und den Betrieb sogenannter *kontrollierter emergenter Systeme* entwickelt werden, d.h. für Software-Ökosysteme, die unter der Kontrolle übergeordneter Instanzen neues Verhalten aus sich selbst hervorbringen. Hierfür wird ein emergentes, verteiltes System als Drei-Schichten-Architektur aufgefasst, bestehend aus einer IoT-, einer Kontroll- und einer DevOps-Schicht.



Diese drei Schichten sind im Schaubild beispielhaft für ein Electric-Grid-Szenario dargestellt. In der IoT-Schicht verhandeln IoT-Ökosysteme, z.B. Häuser, Straßenzüge oder Fabriken ihre Arbeitskonfiguration und Ressourcennutzung. Sie optimieren diese und erzeugen damit Emergenz auf der Ebene der lokalen Komponenten. Die Kontroll-Schicht verwaltet mehrere lokale IoT-Ökosysteme und trifft übergreifende Steuerungsentscheidungen, z.B. dass sich nahegelegene Windkraftanlagen ein- oder ausschalten. Auf der DevOps-Schicht wird das Gesamtsystem überwacht und es werden bspw. neue Energieverbrauchsmuster zur verbesserten Erkennung in die lokalen Umgebungen eingespielt, Sensoren und Geräte aus der Ferne analysiert und fehlerbereinigter Programmcode für Sensoren und Geräte verteilt.

DevOpt entwickelt dabei einen allgemeinen Lösungsansatz, der in einer Vielzahl von Domänen Verwendung finden soll, wie bspw. im Bereich Utilities oder Digital Factory. So könnten europaweit 180 Millionen intelligente Smart-Meter verbaut werden, um den Stromverbrauch zu optimieren. Im Bereich Digital Factory kann ein Multi-Millionen-Markt durch Integration kontrollierter Emergenz mit DevOps bei hochgradig individualisierter Konfiguration und schnelleren Produkt-Update-Zyklen erschlossen werden. Durch existierende Partnernetzwerke und eine kombinierte Open-/Closed-Source-Strategie ergeben sich Multiplikator-Effekte bei Verwertung, Aus- und Weiterbildung.

Bundesministerium für Bildung und Forschung

Fördermaßnahme Resilienz und Ausfallsicherheit
Projektträger DLR PT GI-SUW
Ansprechpartner Dr. Anne Peleg-Rysavy
Tel.: (030) 67055-9646, Anne.Peleg-Rysavy@dlr.de

A.2. Klassendiagramm

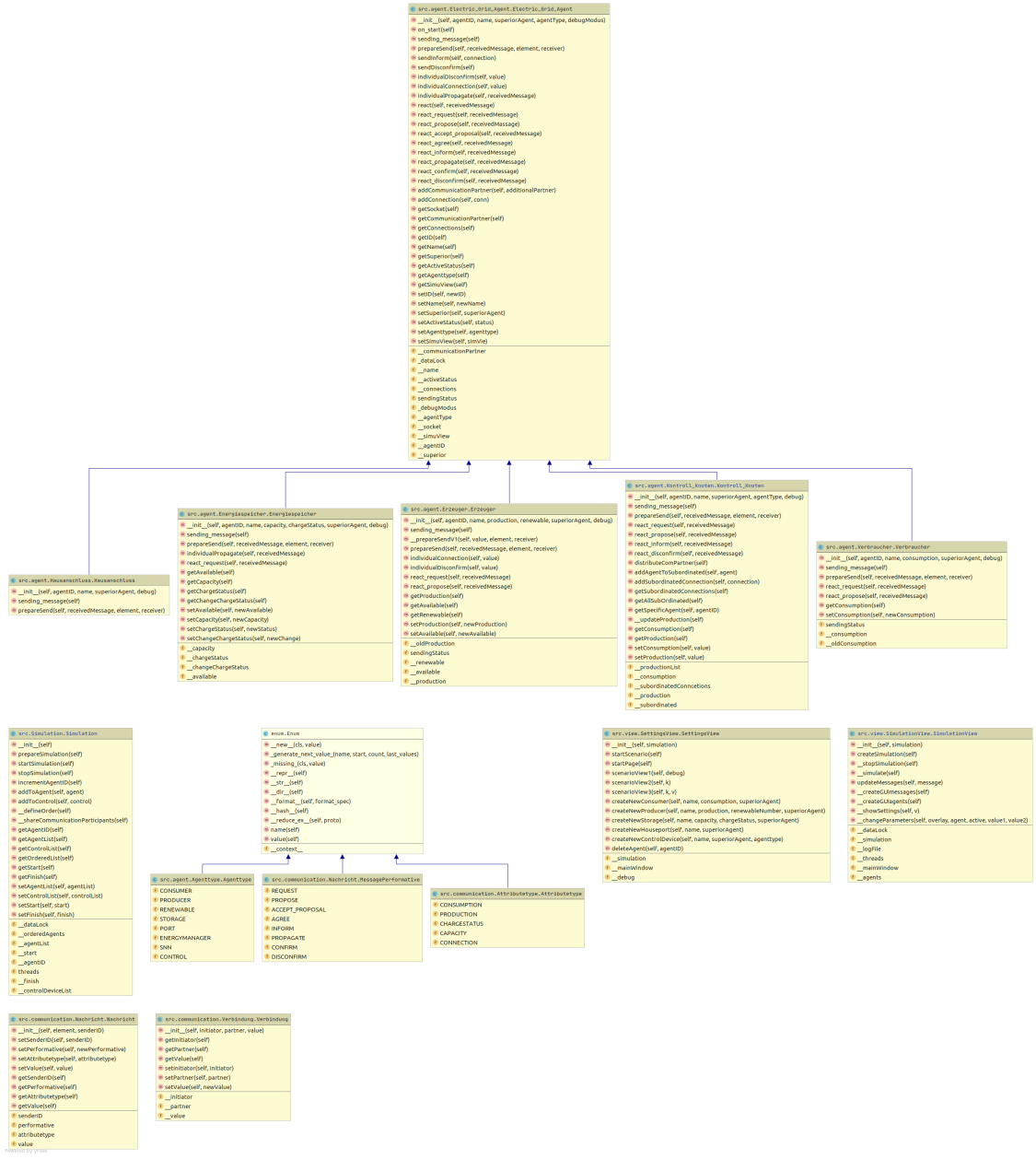


Abbildung A.1.: Klassendiagramm des Simulators. Die Abbildung zeigt die Klassen, Methoden und Attribute des implementierten Simulators. Die Attribute und Methoden vom Zugriffstyp *private* sind in dem Diagramm über ___<Name> gekennzeichnet. Die Kennzeichnung _<Name> steht für den Zugriffstyp *protected*. Das Klassendiagramm veranschaulicht, dass alle Agentenklassen von der Klasse *Electric_Grid_Agent* erben. Dadurch sind die gemeinsamen Charakteristika und Verhaltensweisen eines Agenten in einer Oberklasse gebündelt. Die Bezeichnungen für die Klassen, Methoden und Attribute sind bei beiden prototypischen Implementierungen identisch, sodass das Klassendiagramm für den ersten und den zweiten Prototypen gültig ist.

Abbildungsverzeichnis

Tabellenverzeichnis

Definitions- und Theoremverzeichnis

Abkürzungsverzeichnis

ABMS	Agentenbasierte Modellierung und Simulation
ACL	Agent Communication Language
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
IoT	Internet of Things
JADE	Java Agent Development [Framework]
PADE	Python Agent Development [Framework]
PVA	Photovoltaikanlage
SNN	Sekundäre Schaltanlage (Secondary Substation Node)
TCP	Transmisson Control Protocol
UDP	User Datagram Protocol

Literaturverzeichnis

[BCPR03] Fabio Bellifemine, Giovanni Claire, Agostino Poggi, and Giovanni Rimassa.
Jade: A white paper, 2003.

[DCPF18] João Pedro Dias, Flávio Couto, Ana CR Paiva, and Hugo Sereno Ferreira.
A brief overview of existing tools for testing the internet-of-things.
In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 104–109. IEEE, 2018.

[KdH09] Stamatis Karnouskos and Thiago Nass de Holanda.
Simulation of a smart grid city with software agents.
In *Computer Modeling and Simulation, UKSIM European Symposium on*, EMS '09, pages 424–429. IEEE Computer Society, 2009.

[KJP15] Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti.
Designing a smart city internet of things platform with microservice architecture.
In *2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 25–30. IEEE Computer Society, 2015.

[LBBC19] Cristian Lai, Francesco Boi, Alberto Buschettu, and Renato Caboni.
Iot and microservice architecture for multimobility in a smart city.
In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 238–242. IEEE Computer Society, 2019.

[LGJ19] Isaac Lera, Carlos Guerreo, and Carlos Juiz.
Yafs: A simulator for iot scenarios in fog computing.
IEEE Access, 7:91745–91758, 2019.

[LSL06] Zhengping Li, Cheng Hwee Sim, and Malcolm Yoke Hean Low.
A survey of emergent behavior and its impacts in agent-based systems.
In *2006 4th IEEE International Conference on Industrial Informatics*, pages 1295–1300. IEEE, 2006.

[MN06] Charles M. Macal and Michael J. North.
Tutorial on agent-based modeling and simulation part 2: How to model with agents.
In *2006 Winter Simulation Conference*, pages 73–83. IEEE Computer Society, 2006.

[MN08] Charles M. Macal and Michael J. North.
Agent-based modeling and simulation: Abms examples.

In *2008 Winter Simulation Conference*, pages 101–112. IEEE, 2008.

[MN10] Charles M. Macal and Michael J. North.
Tutorial on agent-based modeling and simulation.
Journal of Simulation, 4:151–162, 2010.

[MR15] Saurabh Mittal and Larry Rainey.
Harnessing emergence: The control and design of emergent behavior in system of systems engineering.
In *Proceedings of the Conference on Summer Computer Simulation*, SummerSim '15, pages 1–10. Society for Computer Simulation International, 2015.

[MSL+19] Lucas Silveira Melo, Raimundo Furtado Sampaio, Ruth Pastôra Saraiva Leão, Giovanni Cordeiro Barroso, and José Roberto Bezerra.
Python-based multi-agent platform for application on power grids.
International Transactions on Electrical Energy Systems, 29(6):e12012, 2019.

[MST17] Christian Müller-Schloer and Sven Tomforde.
Organic Computing - Technical Systems for Survival in the Real World.
Birkhäuser Basel, first edition, 2017.

[ON98] Paul D. O'Brien and Richard C. Nicol.
Fipa—towards a standard for software agents.
BT Technology Journal, 16(3):51–59, 1998.

[PKSL16] T. Pflanzner, A. Kertesz, B. Spinnewyn, and S. Latre.
Mobiotsim: Towards a mobile iot device simulator.
In *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pages 21–27. IEEE Computer Society, 2016.

[PM14] Ashkan Paya and Dan C. Marinescu.
Cloud-based simulation of a smart power grid.
In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*, pages 875–884. IEEE Computer Society, 2014.

[RN03] Stuart Russel and Peter Norvig.
Artificial Intelligence - A Modern Approach.
Prentice Hall, second edition, 2003.

[Rup16] Chris Rupp.
Sophisten,"schablonen für alle fälle (engl.: Patterns for all purposes)," sophist gmbh, 2016, 2016.

[Tan03] Andrew S. Tanenbaum.
Computer Networks.
Pearson Education, fourth edition, 2003.

[TS02] Andrew S. Tanenbaum and Maarten Van Steen.
Distributed Systems: Principles and Paradigms.
Prentice-Hall, 2002.

[VRH04] Peter Van-Roy and Seif Haridi.
Concepts, Techniques, and Models of Computer Programming.
2004.

[WA05] Barry Wilkinson and Michael Allen.
Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.
second edition, 2005.

[WH04] To De Wolf and Tom Holvoet.
Emergence versus self-organisation: Different concepts but promising when combined.
In *International workshop on engineering self-organising applications*, pages 1–15. Springer, 2004.

[WR17] Johannes Weyer and Michael Roos.
Agentenbasierte modellierung und simulation.
TATuP-Zeitschrift für Technikfolgenabschätzung in Theorie und Praxis, 26(3):11–16, 2017.