Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Armin Steudte

Simulation eines ereignisgesteuerten Stromnetzes

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Armin Steudte

Simulation eines ereignisgesteuerten Stromnetzes

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 26.08.2013

Armin Steudte

Thema der Arbeit
Simulation eines ereignisgesteuerten Stromnetzes

Stichworte
Smart Grid, Micro-Grid, Simulation, Complex Event Processing, ereignisgesteuerte Architektur, Planungsalgorithmen, Jboss Drools, Jboss Opta Planner

Kurzzusammenfassung
Der Wandel der Stromnetze von manueller Steuerung zu autonomen intelligenten Netzen wird auf Grund des immer stärkeren Ausbaus der erneuerbaren Energien nötig. Die sich immer schneller ändernden Mengen an nachgefragter und produzierter elektrischer Energie machen eine manuelle Steuerung zusehends schwieriger. Demand Side Management ist dabei ein Weg diesem Problem zu begegnen und das Stromnetz intelligenter zu steuern. Hierfür wird in dieser Arbeit eine Simulationsumgebung sowie eine ereignisgesteuerte Architektur zur Steuerung von Stromnetzen entworfen und realisiert. Das entwickelte System soll aber nicht nur der Steuerung von Stromnetzen dienen, sondern auch Untersuchungen von alternativen Ansätzen sowie die Wissensvermittlung in der Lehre unterstützen. Hierzu wurde ein Ereignismodell des Anwendungsbereichs entwickelt, welches im Rahmen einer ereignisgesteuerten Architektur die Interaktion der Teilsysteme beschreibt. Außerdem wurde das Planungsproblem, Energieverbrauch und -produktion ausgleichen zu müssen, mit Hilfe von Jboss Opta Planner modelliert und verschiedene zur Lösung des Problems benötigte Planungsalgorithmen evaluiert. Das Ergebnis ist eine Simulationsumgebung für Stromnetze, ein Monitoringsystem zur Überwachung der Netzparameter und eine Steuerung, die autonom Maßnahmen zur Netzregulierung plant. Dabei konnte ein Steuerungsprozess mit einer Verzögerung von unter einer Sekunde realisiert werden.

Armin Steudte

Title of the paper
Simulation of an event-driven power grid

Keywords
smart grid, micro-grid, simulation, complex event processing, event-driven architecture, jboss drools, jboss opta planner

Abstract
The rise of renewable energy sources introduces changes in power systems. The power grid has to change from a manual controlled system to an autonomous and intelligent network. The fast changing amount of produced and consumed electric energy makes it nearly impossible to manually control power systems any longer. Demand side management could be one way to solve this problem

and to control the power grid more effectively. In this thesis we are going to develop and implement a simulation environment and an event-driven architecture to control power grids. The developed system purpose is not only to control power systems. It can be also used to evaluate alternative control strategies and to teach students about smart grids. Therefore an event model for a smart grid has been developed, which describes the interaction between subsystems. Furthermore, the planning problem to equate energy production and consumption has been modeled by the use of Jboss Opta Planner. Different planning algorithms have been evaluated for solving the planning problem. The conclusion of this thesis is an integrated simulation environment for power grids, a monitoring subsystem to observe the power grid's parameters and a control process that automatically plans measures to stabilize the power grid. The developed control process is able to plan and execute control measures in less than a second.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Listings

1. Einleitung

Eine neue Generation an Stromnetzen ist durch den zunehmenden Einsatz von Informationstechnologien am entstehen. Bei den sogenannten „Smart Grids“ zieht Informationstechnologie in alle Ebenen des Netzes bis hin zum Verbraucher ein, und ermöglicht so eine besser Kommunikation aller Netzteilnehmer (Farhangi, 2010, S. 19).
Hiermit wird auf die zunehmende verteilter Erzeugung von elektrischer Energie und den steigenden Anteil an erneuerbaren Energiequellen reagiert. Denn durch die Vernetzung der Teilnehmer kann nicht nur die Produktion von elektrischer Energie, sondern auch die Nachfrage gezielt gesteuert werden (Hassan und Radman, 2010, S. 210).
Ein so erweitertes Netz ist in der Lage sich selber zu verwalten und bei Fehlern im System sich selbst zu helfen (Farhangi, 2010; Hassan und Radman, 2010).

1.1. Motivation

Verschiedene Studien kommen übereinstimmend zu dem Schluss, dass der weltweite Energiebedarf in den nächsten 20 bis 30 Jahren weiter steigen wird (Sven Teske, 2007; International Energy Agency, 2012; Bundesanstalt für Geowissenschaften und Rohstoffe, 2012). Bei Sven Teske (2007, S. 44) wird ein Verbrauch von 800.000 Peta Joule pro Jahr in 2050 prognostiziert (vgl. Abbildung 1.1), welcher auch weiterhin zu großen Teilen aus konventionellen Energieträgern gestillt wird.
Die International Energy Agency (2012) und die Bundesanstalt für Geowissenschaften und Rohstoffe (2012, 9) prognostizieren einen Gesamtverbrauch von 18 Gigatonnen Öl-Äquivalent für das Jahr 2035, aber sehen ebenso wie Sven Teske (2007) fossile Energieträger als Hauptlieferant an (vgl. Abbildung 1.2).
Um die negativen Auswirkungen auf die Umwelt zu begrenzen, und somit die Erderwärmung auf 2 °C zu beschränken, sowie auf das absehbare Ende der Erdölreserven zu reagieren (Bundesanstalt für Geowissenschaften und Rohstoffe, 2012, S. 34), hat die Bundesregierung im August 2011 die Gesetze zur Energiewende[1] verabschiedet. Durch unterschiedliche Maßnahmen zur Effizienzsteigerung von Verbrauchern, sowie durch den Ausbau der Stromnetze und der erneuerbaren Energien sollen diese Ziele erreicht werden.

[1] http://www.bundesregierung.de/Content/DE/Artikel/2011/08/2011-08-05-gesetze-energiewende.html

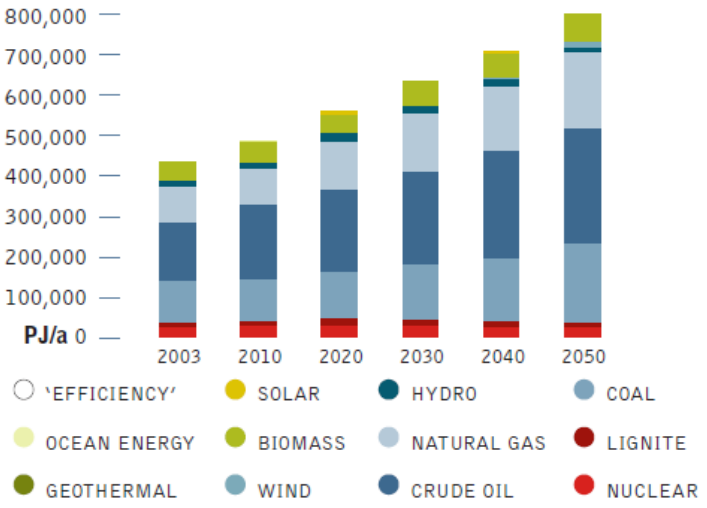

Abbildung 1.1.: Entwicklung des ungebremsten globalen Energieverbrauchs bis 2050 nach Sven Teske (2007)

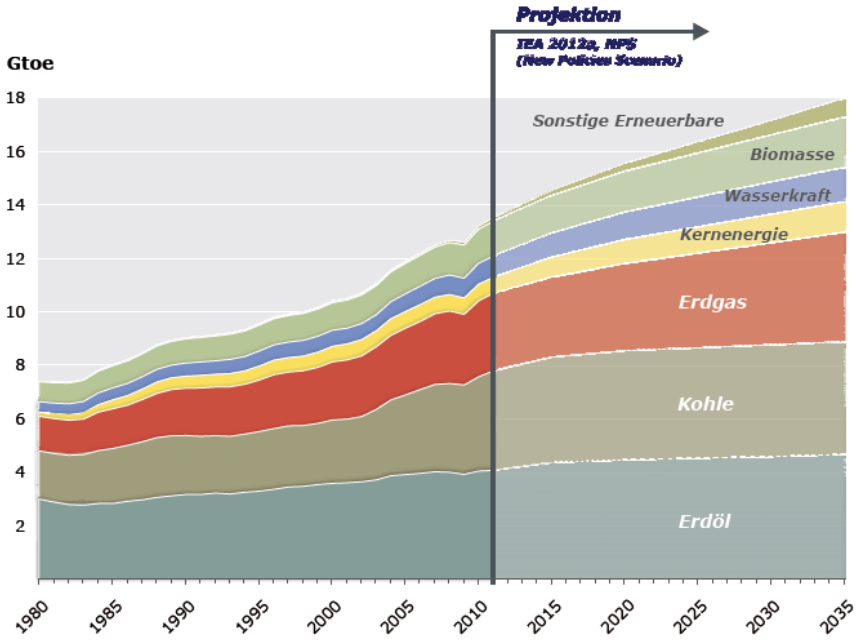

Abbildung 1.2.: Entwicklung des globalen Energieverbrauchs bis 2035 nach International Energy Agency (2012); Bundesanstalt für Geowissenschaften und Rohstoffe (2012)

Durch den daraus resultierenden Ausbau der erneuerbaren Energien kommt es zu neuen Herausforderungen im Bereich der Stromnetze und ihrer Steuerung. Da die produzierte Strommenge in zunehmendem Maße von Umweltfaktoren, wie Windgeschwindigkeit und Sonneneinstrahlung, beeinflusst wird, stellen unvorhergesehene Schwankungen eine Gefahr für die Stabilität der Stromversorgung und des Stromnetzes dar. Dieses zeigt der Bericht der Bundesnetzagentur zum Zustand der leistungsgebundenen Energieversorgung im Winter 2011/12 (Bundesnetzagentur, 2012).

Der Bericht weist darauf hin, dass in kritischen Situationen die vorzuhaltende Regelleistung auf Angebotsseite oftmals nicht mehr ausreicht, um einen stabilen Netzbetrieb zu gewährleisten.
Demand Side Managment (DSM) kann zur Entlastung der Angebotsseite beitragen, indem es eine Steuerung der nachgefragten Energiemenge ermöglicht und so weniger Regelleistung vorgehalten werden muss. Hierdurch können der steigende Bedarf an Kapazitäten gebremst und die Kosten für die Regelleistung, und dadurch die Kosten für elektrische Energieerzeugung, reduziert werden (Short u. a., 2007, S. 1284).

1.2. Vision

Ausgangspunkt dieser Arbeit war die Idee, den Prototypen einer Simulationsumgebung für intelligente Stromnetze zu entwerfen und zu entwickeln. Der Fokus soll hierbei auf einem späteren Einsatz im Bereich der Bildung und Forschung liegen. So könnten mit Hilfe der Umgebung Schülern die Themen erneuerbare Energien sowie die komplexen Abläufe in Stromnetzen näher gebracht werden.
Daher liegt der Schwerpunkt beim Entwurf und der anschließenden prototypischen Umsetzung, auf der Bedienbarkeit, der interaktiven Darstellung und späteren Erweiterbarkeit. So soll sich die Umgebung intuitiv bedienen lassen so dass auch jüngere Personen sie einfach nutzen können. Gleichzeitig soll sich die Simulation in Echtzeit verändern lassen. Die Ergebnisse dieser Veränderungen sollen interaktiv in der Simulation ersichtlich sein.
Das Hauptaugenmerk nimmt hierbei die Entwicklung von Konzepten und nicht die Umsetzung einer realitätsnahen Simulation ein. Wo möglich und sinnvoll soll auf einfache und anschauliche Simulationsmodelle zurückgegriffen werden, um so den Aspekt der Wissensvermittlung in den Vordergrund zu stellen. Gleichwohl sollen sich die Modelle auch einfach gegen neue austauschen und sich so der Grad der Simulationsgenauigkeit anpassen lassen.

1.3. Ziele

Das übergeordnete Ziel der Arbeit ist die Prüfung, inwieweit sich eine ereignisgesteuerte Architektur und der Einsatz von *Complex Event Processing*(CEP) für die Überwachung und Steuerung von intelligenten Stromnetzen eignen. Da noch keine intelligente Strominfrastruktur existiert und eine Testinstallation im Rahmen dieser Arbeit nicht zur Verfügung stand, stellt die Konzeption und Entwicklung einer Simulationsumgebung, wie in Abschnitt 1.2 beschrieben, ein Teilziel dar. Sie

ermöglicht erst die Durchführung der eigentlichen Untersuchungen. Hiervon ausgehend wurden die folgenden Teilziele identifiziert:

Z1) Konzeption und Entwicklung einer Simulationsumgebung für Micro-Grids.

Z2) Entwurf und Realisierung eines Monitoringsystems zur Überwachung des Netzzustandes.

Z3) Durchführung von Untersuchungen über die Eignung der Architektur zur automatisierten Steuerung des Netzes mittels DSM.

1.4. Hypothesen

Ausgehend von den in den Abschnitten 1.1, 1.2, und 1.3 genannten Punkten, liegen der Arbeit Hypothesen zu Grunde, welche in ihrem Verlauf überprüft bzw. widerlegt werden sollen. Dabei handelt es sich um die folgenden zentralen Hypothesen:

H1) Eine ereignisgesteuerte Architektur ermöglicht die Konzeption und Entwicklung eines zuverlässigen und robusten Demand Side Managements.

H2) Mit einem ereignisgesteuerten und kommunikationsbasierten Ansatz lässt sich ein stabiler Netzbetrieb gewährleisten.

H3) Eine autonome und automatisierte Steuerung des Netzes ist auf Basis der betrachteten Architektur möglich.

1.5. Zusammenfassung

Motiviert durch die in Veränderung begriffene Art der Stromerzeugung und den damit einhergehenden Umbau der Stromnetze, soll im Rahmen dieser Arbeit eine ereignisgesteuerte Architektur für DSM und eine ereignisgesteuerte Simulationsumgebung für Micro-Grids entworfen, realisiert und evaluiert werden.
Hierbei sollen ausgehend von der Vision des Einsatzes einer Simulationsumgebung für Smart Grids zum Zweck der Wissensvermittlung, die in Abschnitt 1.3 genannten Ziele verfolgt und die Hypothesen aus Abschnitt 1.4 überprüft werden.

2. Grundlagen

In diesem Kapitel soll zunächst im Abschnitt 2.1 auf die zugrundeliegenden physikalischen Zusammenhänge, sowie den Aufbau und die Steuerung von Stromnetzen eingegangen werden. Darauf aufbauend wird aufgezeigt wie und warum Informatik und Stromnetze verbunden werden sollen, um im Anschluss den Begriff des *Demand Side Managements* (DSM) und die unter ihm formierten Aspekte und Techniken zu betrachten. Abschließend findet die Einordnung des in dieser Arbeit zu entwickelnden Systems in die Aspekte des DSM statt.

2.1. Technische Grundlagen

2.1.1. Physikalische Größen

Diese Arbeit befasst sich, wenn gleich auf einem abstrakten Niveau und vornehmlich aus Sicht der Informatik, mit elektrischen Systemen und deren Zusammenspiel. Daher sollen in diesem Abschnitt die wesentlichen physikalischen Größen und Vorgänge erläutert werden. Hierbei wird die Darstellung auf diejenigen Bereiche, die in dieser Arbeit von Bedeutung sein werden, beschränkt.

Die wichtigsten Basiseinheiten in der Elektrik sind die elektrische Spannung U und der elektrische Strom I. Die Spannung ist der Ladungsunterschied zwischen zwei Punkten und wird in Volt $[\mathrm{V}]$ angegeben. Sie ruft den elektrischen Strom, welcher den Fluss an negativer Ladung bezeichnet, hervor. Dieser wird in Ampere $[\mathrm{A}]$ angegeben (Blume, 2008; Grehn u. a., 1998; Schommers, 2011).

Im Rahmen dieser Arbeit werden die verschiedenen Netzteilnehmer und das Stromnetz jedoch nicht auf der Ebene von Spannung und Strom simuliert bzw. betrachtet. Die Betrachtung erfolgt auf einer höheren Ebene über die elektrische Leistung P und die elektrische Arbeit W (oft auch als elektrische Energie bezeichnet, vgl. hierzu Schommers (2011, 65-68)).
Die elektrische Leistung ist das Produkt von Spannung und Strom, also $P = U * I$ bei Gleichstrom. Die Einheit der elektrischen Leistung ist das Watt, und ist definiert als $1\,\mathrm{W} = 1\,\mathrm{V} * 1\,\mathrm{A}$. Die elektrische Leistung bezeichnet dabei die zu einem Zeitpunkt nachgefragte bzw. gelieferte elektrische Energie, welche im Gegensatz zu Spannung und Strom in der Lage ist reale Arbeit zu verrichten (Blume, 2008, 7).
Darauf aufbauend, ist die elektrische Arbeit die Leistung P, die über einen bestimmten Zeitraum t gewirkt hat (Schommers, 2011, 67). Als Formel ausgedrückt ist die elektrische Arbeit definiert als

$W = P * t = U * I * t$ mit der Einheit Wattsekunde $[\mathrm{W\,s}]$.

Als Basiseinheiten für den weiteren Verlauf der Arbeit, werden für die elektrische Leistung $[\mathrm{W}]$ und für die elektrische Arbeit $[\mathrm{W\,s}]$ gewählt.

2.1.2. Stromnetzaufbau und -steuerung

Aufbau des Stromnetzes

Stromnetze sind, ähnlich wie moderne Softwaresysteme, in verschiedene Ebenen unterteilt. Diese Ebenen unterscheiden sich dabei in der eingesetzten Spannung und in der Entfernung über die sie den Strom transportieren. Die Abbildung 2.1 stellt sie zur Verdeutlichung schematisch dar.
Traditionell wurde der Strom in Großkraftwerken erzeugt und mit Hilfe der Übertragungs- und Verteilnetze zu den Verbrauchern geleitet. Während des Transports wird der Strom auf verschiedene Spannungsstufen transformiert um Übertragungsverluste zu minimieren oder die Spannung an das Niveau des Abnehmers anzupassen. Durch die erneuerbaren Energien, und die durch sie zunehmende dezentrale Energieerzeugung, findet die Erzeugung von elektrischer Energie zunehmend auf allen Netzebenen statt. Somit muss sich der Stromfluss nicht zwangsläufig über alle Ebenen des Netzes erstrecken.

Steuerung des Stromnetzes

Bei der Steuerung von Stromnetzen werden verschiedene Qualitätskriterien verfolgt. Laut El-Hawary (2008, 305) handelt es sich dabei um die Kriterien:

- Sicherheit
- Qualität
- Zuverlässigkeit
- Wirtschaftlichkeit

In der Realität steht die Sicherheit von Personen, Anlagen und der Umwelt an erster Stelle. Die Simulation orientiert sich in dieser Arbeit jedoch ausschließlich am Kriterium der Qualität. Weitere Eigenschaften, wie Zuverlässigkeit und Wirtschaftlichkeit, können zu späteren Zeitpunkten in die Untersuchungen mit einbezogen werden. Das zentrale Augenmerk liegt darauf eine qualitativ hochwertige Stromversorgung durch den Einsatz von *Demand Side Management*(DSM) sicherzustellen.

Der Aspekt der Qualität in der Stromversorgung kann aus zwei Blickwinkeln betrachtet werden (Kundur u. a., 1994, 581):

1. Frequenzstabilität

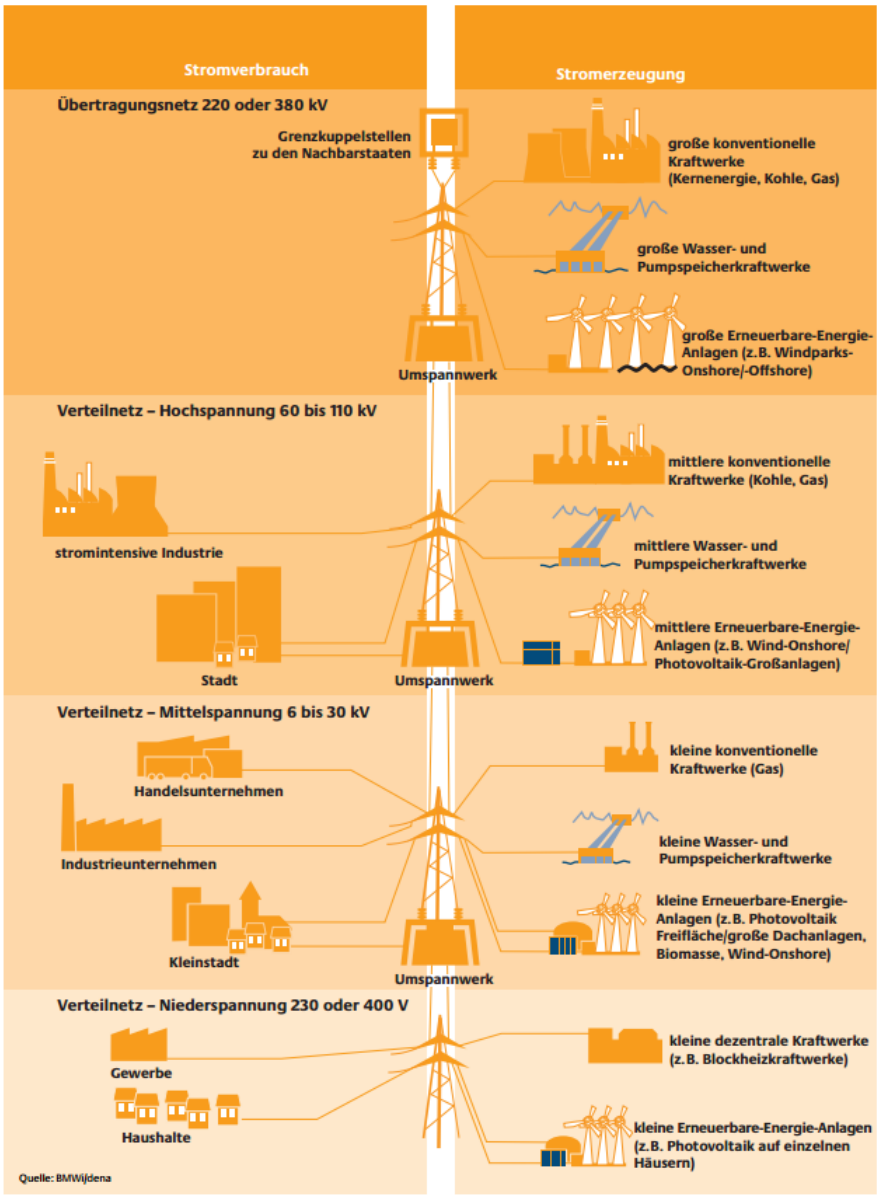

Abbildung 2.1.: Topologie des deutschen Stromnetzes (Deutsche Energie-Agentur GmbH, 2012)

2. Spannungsstabilität

Stabilität bedeutet in beiden Gebieten, dass gewisse Wertebereiche nicht oder nur kurzzeitig überschritten werden dürfen, da es sonst zu Schäden an Anlagen, Geräten oder zu einem kompletten Netzausfall kommen kann. Während bei der Frequenzstabilität die *Wirkleistung* P (active power), also die real verrichtet Arbeit (Blume, 2008, 11), im Vordergrund steht, befasst sich die Spannungsstabilität mit der *Blindleistung* Q (reactive power). Wie bereits in Abschnitt 2.1.1 erwähnt, wird in dieser Arbeit von elektrischer Spannung und Strom abstrahiert, so dass sich bei der Betrachtung des Qualitätsaspekts auf die Frequenzstabilität und damit die Wirkleistung P beschränkt wird. Dabei wird sich an dem in Short u. a. (2007); Kundur u. a. (1994) beschriebene Verfahren des *Load-Frequency Control* (LFC) orientiert.

Frequenzstabilität

In Stromnetzen wird die Frequenz als Mittel zur Synchronisation und zur Steuerung von zeitabhängigen Abläufen genutzt. Hierzu ist es erforderlich, dass die Frequenz über den gesamten zeitlichen Verlauf nahezu konstant gehalten wird (Kundur u. a., 1994, 581).
Als Standards für Netzfrequenzen haben sich auf der Welt $50\,\mathrm{Hz}$ und $60\,\mathrm{Hz}$ etabliert. In Europa werden Stromnetze mit einer Frequenz von $50\,\mathrm{Hz}$ betrieben, während sich in Nordamerika $60\,\mathrm{Hz}$ durchgesetzt haben. Auch alle anderen Länder nutzen entweder $50\,\mathrm{Hz}$ oder $60\,\mathrm{Hz}$ (Blume, 2008, 9). In dieser Arbeit wird mit einer Normfrequenz von $f = 50\,\mathrm{Hz}$ gearbeitet.

Beeinflusst wird die Netzfrequenz durch die Differenz zwischen angebotener und nachgefragter Wirkleistung. Anders gesagt muss für eine stabile Netzfrequenz die Menge an angebotener und nachgefragter Energie zu jedem Zeitpunkt ausgeglichen sein (Blume, 2008, 139). Besteht eine Unterdeckung (höherer Energiebedarf als -produktion) so fällt die Frequenz des Netzes, bei einer Überdeckung (höhere Energieproduktion als -bedarf) steigt sie an (Short u. a., 2007, 1). Somit stellt die Frequenz bzw. der Frequenzverlauf einen Indikator für die Differenz zwischen angebotener und nachgefragter Energiemenge innerhalb des Stromnetzes dar.

Weicht die Netzfrequenz von ihrem Sollwert ab, muss entweder die Produktionsmenge (heutiger Ansatz), der Energiebedarf (Demand Side Management) oder beide Seiten angepasst werden. Es gibt verschiedene Vorgaben, welche Abweichung von der Normfrequenz noch als akzeptabel angesehen werden und ab wann gehandelt werden muss (für ausführliche Informationen sei hier auf die Seite der ENTSO-E[1] und insbesondere auf das Operation Handbook[2] verwiesen).
In der Gesamtschau kann angenommen werden, dass eine Abweichung von $\pm 0{,}02\,\mathrm{Hz}$, also der

[1] https://www.entsoe.eu/
[2] https://www.entsoe.eu/fileadmin/user_upload/_library/publications/entsoe/Operation_Handbook/Policy_1_final.pdf

Frequenzbereich von 49,98 Hz bis 50,02 Hz, im Normbereich liegt. Kurzfristig werden Abweichungen von bis zu ±0,2 Hz toleriert. Diese Werte bzw. Wertebereiche sollen im Verlauf als Kriterien für den Test des Gesamtsystems genutzt werden.

2.2. Informatik und Stromnetze

Das Stromnetz besteht seit über 100 Jahren in seiner jetzigen Form nahezu unverändert. Durch die sich ändernden Anforderungen und die Entwicklung neuer Technologien, sind das Netz und die ihm zugrundeliegenden Konzepte stetig gealtert. Daher ist eine Überholung des selbigen nötig geworden (Gungor u. a., 2011, S. 529).

Während dieses Zeitraums entstanden neue Technologien in den Bereichen Informatik und Telekommunikationsinfrastruktur. Sie werden zukünftig gemeinsam das Fundament für ein erneuertes Stromnetzes bilden. Ein um eine Telekommunikations- und Informationsinfrastruktur erweitertes Stromnetz wird als „Smart Grid“ bezeichnet (Farhangi, 2010, S. 20).
Ein so runderneuertes Stromnetz ist für die zukünftigen Herausforderungen in Bezug auf die Integration erneuerbarer Energien und einer zuverlässigen sowie qualitativ hochwertigen Stromversorgung besser gerüstet (Hassan und Radman, 2010). Außerdem können so neue Wege in Bezug auf eine verteilte Energieerzeugung, automatisches Load Balancing und die Verringerung von Übertragungsverlusten im System beschritten werden (Gungor u. a., 2011, S. 529).

2.2.1. Neuerungen

Existing Grid	Intelligent Grid
Electromechanical	Digital
One-Way Communication	Two-Way Communication
Centralized Generation	Distributed Generation
Hierarchical	Network
Few Sensors	Sensors Throughout
Blind	Self-Monitoring
Manual Restoration	Self-Healing
Failures and Blackouts	Adaptive and Islanding
Manual Check/Test	Remote Check/Test
Limited Control	Pervasive Control
Few Customer Choices	Many Customer Choices

Abbildung 2.2.: Gegenüberstellung der Eigenschaften des aktuellen Stromnetzes und des Smart Grids (Farhangi, 2010)

Abbildung 2.2 stellt die wichtigsten Eigenschaften von altem und neuem Netz gegenüber. Hieraus lassen sich auch die wesentlichen Neuerungen und Möglichkeiten ableiten, welche durch die Transition hin zu einem Smart Grid ermöglicht werden.
Zu diesen gehört vor allem die Digitalisierung des Netzes, wie sie bereits in den Telekommunikationsnetzen stattgefunden hat. Schalt- und Steuervorgänge erfolgen nicht mehr elektromechanisch, sondern werden durch digitale Signale getätigt. Die Übermittlung von Befehlen und Daten ist nach dem Umbau nicht nur von der Leitstelle zu den angeschlossenen Einheiten möglich, sondern die Netzbetreiber sind dann auch in der Lage Daten in Echtzeit aus dem Netz zu erhalten. Die Einführung einer Zweiwege-Kommunikation, welche erst durch die Allgegenwärtigkeit der Kommunikationsinfrastruktur (DSL, Mobilfunk, Powerline uvm.) ermöglicht wird, trägt so zu einer genaueren und effizienteren Steuerung sowie Überwachung des Stromnetzes bei. In der bisherigen Form ist dies nur bedingt möglich.
Bei den Punkten Überwachung und Steuerung steht aber nicht nur die manuelle Steuerung im Vordergrund, vielmehr wird das Netz durch den Einsatz von Informationstechnik in die Lage versetzt, sich selbst zu überwachen und zu steuern. Es nimmt seinen Zustand wahr und kann automatisch Aktionen ergreifen, um einen stabilen Betrieb zu gewährleisten und kritische Situationen zu vermeiden (*Self-Healing*).

2.2.2. Micro-Grids

Werden die erweiterten Überwachungs- und Steuerungsmöglichkeiten mit einer dezentralen Energieerzeugung, z.B. durch Blockheizkraftwerke oder Biomasse, verbunden, so kann eine weitere Strukturierung des Smart Grids durch sogenannten Micro-Grids erfolgen (Hatziargyriou u. a., 2007, S. 79).
Bei Micro-Grids handelt es sich um kleine autonome „Inseln“ innerhalb des großen Stromnetzes. Sie sind im Bereich der Verteilnetze angesiedelt und können auch komplett losgelöst vom übergeordneten Stromnetz funktionieren, indem sie sich selbst verwalten. Die Abkopplung eines Smart Grids kann bei Problemen sinnvoll sein, um ein Übergreifen auf das gesamte Stromnetz zu verhindern (Huang u. a., 2011, S. 206).
Micro-Grids werden auf Grund ihrer Abgeschlossenheit als eine Möglichkeit betrachtet das vorhandene Stromnetz sukzessive hin zu einem Smart Grid zu entwickeln. Bei diesem Vorgehen können zuerst einzelne Teilbereiche des Netzes auf- und umgerüstet werden, ohne das die Funktionsfähigkeit des Gesamtsystems beeinträchtigt wird (Farhangi, 2010; Huang u. a., 2011).
Im Rahmen dieser Arbeitet bietet sich die Betrachtung von Micro-Grids, neben dem Aspekt der Abgeschlossenheit, auch wegen der geringeren Anzahl an zu betrachtenden Netzteilnehmer an. Die Komplexität der Simulation lässt sich so begrenzen. Aus diesem Grund werden innerhalb dieser Arbeit alle Konzepte im Rahmen eines Micro-Grid-Szenarios betrachtet und simuliert.

2.2.3. Zusammenfassung

Die Kombination von elektronischer Schalttechnik und moderner Informations- und Kommunikationstechnik erlaubt es das existierende Stromnetz um die Fähigkeit zur Überwachung und Steuerung in nahezu Echtzeit zu erweitern. Diese Fähigkeit wird das Netz zukünftig in die Lage versetzen sich selbst zu überwachen und zu steuern.
Micro-Grids nutzen diese neue Fähigkeit in Verbindung mit einer dezentralen Energieerzeugung und können so dem vorhanden Stromnetz eine völlig neue Struktur geben.
Alle diese Entwicklungen sollen dazu beitragen das Stromnetz effizienter zu machen, indem Verluste vermieden und Überkapazitäten abgebaut werden.

2.3. Demand Side Management

Um eine ausgeglichene Differenz zwischen angebotener und nachgefragter Energie zu erhalten, können die Menge an produzierter elektrischer Energie, die Menge an nachgefragter Energie oder beide Seiten angepasst werden. In der Vergangenheit wurde stets die Produktionsmenge an die Nachfrage angepasst, da die Angebotsseite bisher nicht von den Betreibern gesteuert werden konnte (Strbac, 2008, 3). Dieses führt in Verbindung mit dem Einsatz von erneuerbaren Energien dazu, dass große Mengen an Regelenergie, in Form von konventionellen Kraftwerksreserven, vorgehalten werden müssten, um eventuelle Produktionsausfälle kompensieren und jederzeit die Spitzennachfrage bedienen zu können (Haubrich, 2008, 4).

Unter *Demand Side Management* (DSM) versteht man im Allgemeinen alle Maßnahmen die auf die Veränderung des Lastverhaltens von elektronischen Verbrauchern in Haushalten und der Industrie abzielen (Gellings, 1985, 1). Im deutschsprachigen Raum ist DSM unter dem Begriff der „Laststeuerung“ schon seit längerer Zeit bekannt (Serafin von Roon, 2010, 1).
Laut Kupzog (2006, 2) gehören zu den Maßnahmen nicht nur automatische Verfahren, wie sie im Zentrum der Untersuchungen in dieser Arbeit stehen, sondern auch die manuelle Regelung z.B. durch das Tätigen von Telefonanrufen für die Breitstellung von Minutenreserven. Auch die Beeinflussung des Nutzerverhaltens durch Anreize, wie besondere Stromtarife z.B. für Nachtspeicherheizungen (Strbac, 2008, 14), können hierunter eingeordnet werden (Gellings, 1985, 1468). Andere Definitionen, wie die von Klobasa (2006, 3), sehen den Schwerpunkt eher bei der zeitlichen Entkopplung von Stromangebot und -nachfrage und konzentrieren sich auf die Maßnahme des *Load Shiftings*.
Generell wird eine Vielzahl an Maßnahmen und Techniken zur Beeinflussung der Angebotsseite unter dem Begriff zusammengefasst. Daher soll im folgenden Abschnitt eine genauere Einordnung und Betrachtung der einzelnen Möglichkeiten erfolgen.

2.3.1. Demand Side Management Bereiche und Techniken

Die Tabelle in Abbildung 2.3 zeigt eine Übersicht über die verschiedenen Aspekte des DSM. Dabei wird auch auf die dem Aspekt zugrundeliegende Motivation und die verwendeten Operationen eingegangen.

	Efficiency and conservation (daily)	Peak load management (daily)	Demand response (dynamic event driven)
Motivation	▪ Conservation ▪ Environmental protection	▪ Time Of Use (TOU) savings ▪ Peak demand charges ▪ Grid peak	▪ Price ▪ Reliability ▪ Emergency
Design	▪ Efficient shell, equipment & systems	▪ Low Power Design	▪ Dynamic control capability
Operations	▪ Integrated system operations	▪ Demand limiting ▪ Demand shifting	▪ Demand shedding ▪ Demand shifting ▪ Demand limiting

Abbildung 2.3.: Bereiche des Demand Side Management (Kupzog, 2008; Kiliccote u. a., 2006)

Am Anfang der Begriffsbildung, in den 1980ern und 1990ern, verstand man unter DSM die Steigerung der Effizienz des Stromnetzes (Efficiency and conservation) sowie die eigentliche Laststeuerung zur Reduktion der Spitzenlast (Peak load management) auf Seiten der Energienachfrage (Gellings, 1985, 1469). Durch den Fortschritt der Informationstechnologie und die Einführung besonderer Tarifmodelle zur Schaffung von Anreizen zur Lastminderung, kam in den folgenden Jahrzehnten der Bereich der dynamischen Steuerung der Nachfrage (Demand Response) hinzu (Kiliccote u. a., 2006, 3).

Efficiency and conservation

Dieser Aspekt umfasst alle Maßnahmen zur Steigerung der Energieeffizienz von Geräten und Anlagen beim Verbraucher, in der Industrie aber auch den Netzbetreibern. Durch finanzielle Anreize und Programme soll der ständig wachsende Energiebedarf und die dadurch nötige Aufstockung der Produktionskapazitäten eingeschränkt werden. Auslöser hierfür war unter anderem die Öl-Krise in den 1970ern (Gellings, 1985, S. 1469).

Peak load management

Ziel dieses Aspekts ist es Spitzen im Lastverhalten zu reduzieren und so die Menge an vorzuhaltenden Produktionskapazitäten und damit der benötigten Regelenergie zu verringern. Hierzu werden durch tarifliche Anreize, vor allem in der Industrie, Last aus den Spitzenzeiten in Zeiträume mit weniger Nachfrage verlagert (*Demand Shifting*). Eine andere Möglichkeit besteht in der Abschaltung von Lasten (*Demand Limiting*), wenn die Nachfrage einen bestimmten Grenzwert überschreitet. Dieses wird durch vertragliche Zusicherungen und finanzielle Anreize unterstützt (Kupzog, 2008; Kiliccote u. a., 2006).

Demand response

Bei Demand Response findet eine dynamische Steuerung der Nachfrage statt. In Reaktion auf bestimmte kritische Ereignisse oder Situationen wird der Kunde angewiesen bestimmte Aktionen auszuführen, die zuvor vertraglich vereinbart wurden (Kupzog, 2008, 28). Neben den bereits genannten Techniken *Demand Shifting* und *Demand Limiting*, welche einer statischen Natur sind, wird zusätzlich *Demand Shedding* eingesetzt. Hierbei wird dynamisch die Last einzelner Verbraucher, je nach aktuellem Bedarf, kurzzeitig verringert (Kiliccote u. a., 2006, 3).

Eingesetzte Techniken

Einige Techniken die beim DSM zum Einsatz kommen, wurden in den vorangegangenen Abschnitten bereits vorgestellt:

- *Demand Limiting*
- *Demand Shedding*
- *Demand Shifting*

Neben diesen in Kupzog (2008) und Kiliccote u. a. (2006) beschriebenen Techniken, können bei Gellings (1985) und Strbac (2008) weitere Maßnahmen gefunden werden. Diese ergänzen die oben beschriebenen Maßnahmen oder stellen konkrete Ausprägungen von ihnen dar. Daher soll im Folgenden die Liste um die Punkte aus Gellings (1985) und Strbac (2008) ergänzt werden, so dass danach eine Ordnung des in dieser Arbeit zu entwickelnden Systems erfolgen kann.

Gellings (1985) subsumiert unter dem Begriff des DSM bekannte Techniken zur Lastverteilung (vgl. Abbildung 2.4). In ihnen sind auch die bereits vorgestellten Maßnahmen, unter anderer Bezeichnung, enthalten:

- Peak Clipping = Demand Limiting
- Flexible Load Shape = Demand Shedding

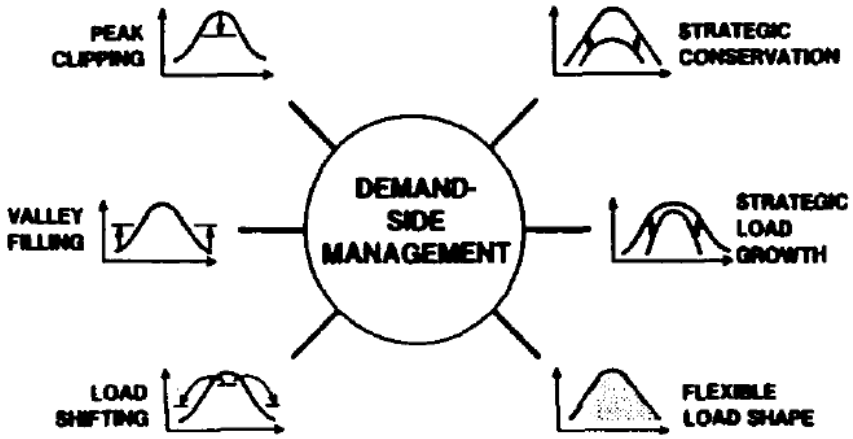

Abbildung 2.4.: Techniken des Demand Side Management nach Gellings (1985)

- Load Shifting = Demand Shifting

Diese Techniken haben das Ziel der Lastverringerung gemeinsam. Während die Maßnahmen Valley Filling und Stategic Load Growth der „künstlichen“ Erzeugung von Last dienen. Startegic Conservation hat dabei eine Sonderstellung und entspricht dem unter „Efficiency and Conservation“ beschriebenen Vorgehen.

Strbac (2008) zählt zumeist konkrete Umsetzungen der Maßnahmen, wie Nachtspeichertarife für das *Demand Shifting*, auf. Besonders interessant dabei aus Sicht der Informatik sind die Punkte *Direct-load Control* und *Frequency Regulation*.
Ersteres soll in dieser Arbeit simuliert und mit der zu entwickelnden Architektur dazu genutzt werden, um ein sich selbst steuerndes Stromnetz zu entwickeln. In den zweiten Bereich fällt die Methode des *Dynamic Demand Control*. Die in dieser Arbeit gewonnenen Ergebnisse sollen abschließend mit den Ergebnissen von Arbeiten aus dem Bereich des *Dynamic Demand Control* verglichen werden. Auf diesem Weg soll ein Vergleich zwischen kommunikationsbasierten Ansätzen und Ansätze, die keine Kommunikation zwischen Geräten einsetzen, erreicht werden. Sowohl *Direct-load Control* als auch *Frequency Regulation* werden eher als übergeordnete Punkte gesehen, und daher nicht zu den konkreten Techniken gezählt.
Zum Abschluss gibt Tabelle 2.1 eine zusammenfassende Übersicht über die unterschiedliche Techniken und die Kategorie in die sie fallen.

2.3.2. Einordnung der Arbeit

Diese Arbeit ordnet sich vom Schwerpunkt her in den Bereich des *Demand Response* ein. Die Fähigkeit des zu entwickelnden Systems, dynamisch das Stromnetz zu steuern und auf neue Situationen zu reagieren, entspricht genau dem „Dynamic Control“-Gedanken. Auch wird sowohl bei *Demand Response*,

Load Reduction	**Load Enhancement**
• *Demand Limiting* (Peak Clipping) • *Demand Shedding* (Flexible Load Shape) • *Demand Shifting* (Load Shifting)	• *Valley Filling* • *Stategic Load Growth*

Tabelle 2.1.: Übersicht über im Demand Side Management eingesetzte Techniken

als auch in der Arbeit, besonderes Augenmerk darauf gelegt, dass das System einen verlässlichen Betrieb ermöglicht und auch kritische Situationen meistern kann.

Bei den verwendeten Techniken sollen sowohl Techniken aus dem Bereich Load Reduction, als auch aus dem Bereich Load Enhancement zum Einsatz kommen. Je nach Situation soll das System sowohl Lasten ein- und ausschalten, drosseln oder erst zu bestimmten Zeitpunkten einschalten können. Es stehen ihm also alle Möglichkeiten des Lastmanagements zur Verfügung, und sollen ausdrücklich auch genutzt werden, um einen möglichst effizienten und zuverlässigen Betrieb des Stromnetzes sicherzustellen.

3. Verwandte Arbeiten

In diesem Kapitel werden andere Arbeiten aus den Bereichen der Simulation von Smart Grids und dem Teilbereich der Simulation von Demand Response vorgestellt. Außerdem werden die jeweils in den Arbeiten gewählten Ansätze mit dem in dieser Arbeit verfolgten Ansatz verglichen und ihre Unterschiede herausgearbeitet.

Der Schwerpunkt dieser Arbeit liegt nicht direkt in der Simulation des Smart Grids. Diese wird zu dem Zweck eingesetzt, eine Infrastruktur zu simulieren, auf deren Basis unterschiedliche Steuerungsstrategien untersucht und erprobt werden können.
Da die realitätsnahe Simulation von Stromnetzen und den beteiligten Abläufen ein sehr aufwändiges Unterfangen ist, stellt dieses keinen Kernaspekt der Arbeit. So weit es möglich ist wird auf einfache Modelle zurückgegriffen und von komplexen Zusammenhängen abstrahiert.
Der Schwerpunkt liegt auf der prototypischen Umsetzung der Teilsysteme, sowie auf der Entwicklung und der Durchführung von Untersuchungen von Strategien zur reaktiven Steuerung des Netzes. Da der Simulationsaspekt hierbei weiterhin eine wichtige Rolle spielt, sollen im Verlauf dieses Kapitels auch verwandte Ansätze aus diesem Bereich vorgestellt und verglichen werden.

3.1. Smart Grid Simulation

3.1.1. GridLAB-D

GridLAB-D ist eine modulare Open Source Simulations- und Modellierungsumgebung für Stromnetze. Sie wird am Pacific Northwest National Laboratory in Kooperation mit dem amerikanischen Energieministerium entwickelt und steht der breiten Öffentlichkeit seit Oktober 2008 zur Verfügung. Federführend für die Entwicklung von GridLAB-D zeichnen sich D. P. Chassin, K.Schneider und C. Gerkensmeyer (Chassin u. a., 2008).

Projektziele

GridLAB-D bietet eine Lösung zur Modellierung und Simulation von Stromnetzen über alle Netzebenen hinweg an. Mit GridLAB-D können unterschiedlichste Aspekte untersucht und überprüft werden. Diese Lösung zeichnet sich durch die einfache Integration von externen Analysewerkzeugen für Stromnetze aus.

Um diese Vielfalt an Möglichkeiten realisieren zu können, besitzt GridLAB-D eine modulare Architektur, so lassen sich neue Simulations- und Betriebsmodelle integrieren (Chassin und Widergren, 2009, S. 3,4). Standardmäßig unterstützt GridLAB-D die folgenden Simulationsfunktionalitäten:

- Simulation und Steuerung des Stromflusses.
- Simulation des Endbenutzerequipments und dessen Verhalten.
- Die Überwachung der Eigenschaften von simulierten Objekten.

Funktionsweise

Um die beschriebenen Ziele umsetzen zu können, wird ein agenten-basierter Ansatz genutzt. In Verbindung mit einem nicht näher beschriebenen hocheffizienten Algorithmus, wird jeder Bestandteil des Netzes als Agent modelliert. Während der Simulation berechnen alle Agenten, unabhängig von einander, ihren neuen Zustand. Dieser ist von ihrem vorherigen Zustand und der aktuellen Simulationszeit abhängig (Chassin u. a., 2008, S. 1).
Der agenten-basierte Ansatz ermöglicht, in Verbindung mit einer effizienten Systemarchitektur, ermöglichen einen hohen Detailgrad bei der Simulation. Zusätzlich trägt dieser Ansatz dazu bei, dass GridLAB-D besonders von dem Einsatz auf Mehrkern- und Multiprozessorsystemen profitiert.

Bewertung

Die Idee die hinter GridLAB-D kommt der in dieser Arbeit beschriebenen Vision einer Simulationsumgebung sehr nahe. In einigen Punkten, wie Detailgrad und Realitätsnähe, ist sie der hier beschriebenen Lösung überlegen. Als konkretes Beispiel sei hier die Fähigkeit zur Simulation mehrerer Stromflüsse angeführt (vgl. Schneider u. a. (2009)).
Dennoch wurde sich bewusst gegen den Einsatz von GridLAB-D und für die Durchführung einer Eigenentwicklung entschieden. Die Gründe hierfür lagen in den Unwägbarkeiten bei der Umstellung der Simulationsmodelle hin zu einer Steuerung mittels Ereignissen. Da GridLAB-D von sich aus keine ereignisgesteuerte Architektur besitzt, ist es sehr wahrscheinlich, dass alle Simulationsmodelle angepasst werden müssten. Dieses ist zwar realistisch möglich, weil GridLAB-D Open Source ist. Der erwartete Aufwand würde aber den Aufwand einer bedarfsgenauen Neuentwicklung deutlich übersteigen. Außerdem besitzt GridLAB-D eine Simulationstiefe und -komplexität, die für die in dieser Arbeit betrachteten Anwendungsfälle nicht benötigt wird und die erforderlichen Erweiterungen verkomplizieren würde.
Schließlich ergab die Bewertung des von GridLAB-D gewählten Ansatzes folgendes Ergebnis:

+ Open Source
+ Ausführliche Dokumentation (Source Code und Tutorials)
+ Große Verbreitung

+ Unterstützt die Entwicklung eigener Module
+ Skalierbarkeit
- Großer Funktionsumfang
- Unübersichtlich

3.1.2. HOMER

Bei Homer handelt es sich um ein Simulationswerkzeug zur Durchführung von Wirtschaftlichkeitsanalysen. Das Akronym Homer steht hierbei für „Hybrid Optimization Model for Electric Renewables“. Es wird seit 1993 am *National Renewable Energy Laboratory* in Colorado entwickelt und befindet sich seit dem unter ständiger Weiterentwicklung. Verantwortlich für die Entwicklung zeichnen P. Lilienthal, T. Lambert und P. Gilman (Lambert u. a., 2006).

Projektziele

Homer unterstützt bei der Durchführung folgender Aufgaben in hybriden *Micropower Systemen* (Givler und Lilienthal, 2005):

- Modellierung
- Simulation
- Evaluierung
- Optimierung

Der Schwerpunkt des Projektes liegt hierbei in der Wirtschaftlichkeitsanalyse und der Optimierung von Netzentwürfen. Als Ergebnis liefert Homer eine Hochrechnung der Kosten über den Lebenszyklus eines Szenarios und macht verschiedene Vorschläge zur Optimierung unter Berücksichtigung von gegebenen Rahmenbedingungen.

Funktionsweise

Im Detail bietet Homer Modelle für die wichtigsten Komponenten eines Stromnetzes an. Das reicht von verschiedenen Kraftwerksarten, sowohl für konventionelle als auch erneuerbare Energien, bis hin zu Verbrauchern wie z.B. Lampen und anderen Haushaltsgeräten.
Mit Hilfe der Modelle können Entwürfe für zu planende Stromnetze erstellt werden. Im Anschluss können diese Szenarien dann simuliert werden. Hierbei ermöglicht es Homer für die Simulation verschiedene Rahmenbedingungen, wie z.B. Wind-/Sonnenverhältnisse, die Entwicklung von Kraftstoffpreisen u.v.m. vorzugeben.
Auf Basis der Rahmenbedingungen und unter kostentechnischen Gesichtspunkten werden dann

verschiedene Abwandlungen des vorher erstellten Szenarios simuliert. Dabei wird in stündlichen Schritten vorgegangen und jede Abwandlung über eine feste Simulationsdauer von einem Jahr durchgerechnet. Auf Basis dieser Daten erfolgt zum Schluss eine Hochrechnung der Kosten über die gesamte Lebensdauer des Netzes.

Simulationsergebnisse

Zum Beginn einer Simulation wird zunächst über die Benutzeroberfläche ein Szenario konfiguriert. Eine Beispielkonfiguration ist in Abbildung 3.1 dargestellt.
Das Beispielszenario beinhaltet zwei Energieerzeuger (ein Dieselgenerator und eine Windkraftanlage), einen Batteriepuffer sowie einen Verbraucher mit 85 kWh/d. Zusätzlich werden noch die Rahmenbedingungen, wie z.B. in diesem Fall die geschätzte Preisentwicklung des Kraftstoffpreises, eingestellt.

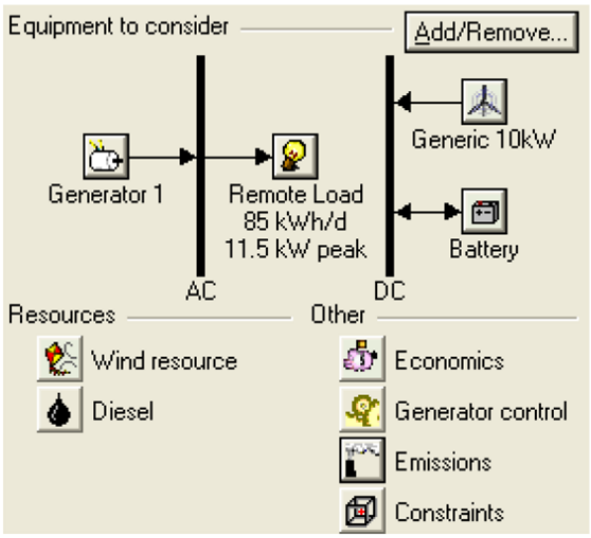

Abbildung 3.1.: Modell eines Beispielszenarios in Homer (Lilienthal u. a., 2005)

Nachdem die Konfiguration abgeschlossen wurde, wird das Szenario simuliert und man erhält einen ähnlichen Optimierungsgraphen wie er in Abbildung 3.1.2 dargestellt ist.

In dem Optimierungsgraphen wird dargestellt, für welche Randbedingung welche Konfiguration des Szenarios am kostengünstigsten ist.

Bewertung

Im Folgenden finden sich zusammengefasst die wesentlichen Vor- und Nachteile des von Homer gewählten Ansatzes:

+ Unterstützung von Hybrid Systems (erneuerbare und konventionelle Energieträger)

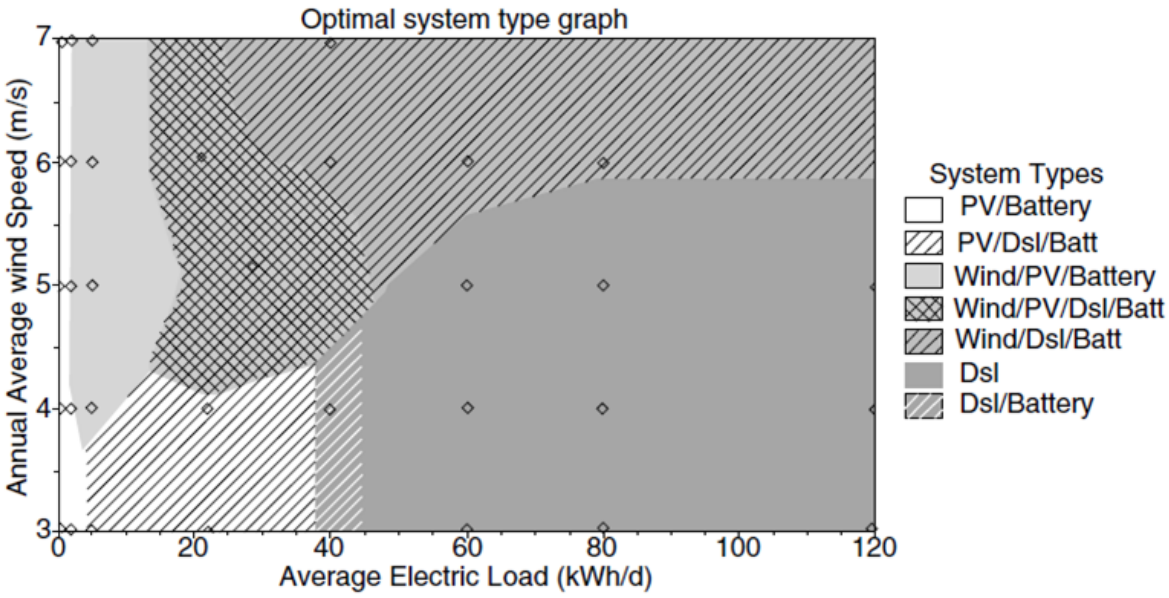

Abbildung 3.2.: Beispiel für das Ergebnis einer Simulation in Homer (Lambert u. a., 2006)

+ Simulation von Micro-Grids möglich
+ Lastkurven zur Beschreibung des elektr. Verhaltens von Komponenten
- Schwerpunkt Kostenoptimierung
- Kontrollstrategien und Szenarien fest vorgegeben
- feste Simulationsdauer
- Technische Darstellung bei der Szenariogestaltung

Besonders im Hinblick auf eine interaktive Simulation und das Experimentieren mit Szenarien, weist Homer einige Einschränkungen auf.
Die festen Vorgaben bezogen auf Simulationsdauer, Simulationsschritte sowie Kontrollstrategien sind für den Zweck der Kostenschätzung und -optimierung sinnvoll, machen aber die Integration neuer Strategien und Ansätze unmöglich und schränken die Möglichkeit zu experimentieren stark ein.
Diese Punkte führen dazu, dass eine weitere Evaluation von Homer nicht zielführend erschien.

3.2. Demand Side Management Simulation

Grenzt man den Kreis der Arbeiten auf die ein, welche sich mit der Simulation von *Demand Side Management* (DSM) befassen, so stößt man hierbei auf die Arbeiten von Short u. a. (2007); Kupzog (2008); Zaidi u. a. (2010); Gudi u. a. (2011); Croft u. a. (2011); Zeilinger (2011). Dabei setzt keine dieser

Arbeiten auf eine ereignisgesteuerte Architektur, um DSM zu realisieren.

Bei genauerer Betrachtung lassen sich die Arbeiten in die Kategorien „kommunikationsbasierend“ und „Dynamic Demand Control“ (DDC) unterteilen (vgl. hierzu Tabelle 3.1).

Kommunikationsbasierend	**Dynamic Demand Control**
• Zaidi u. a. (2010) • Gudi u. a. (2011) • Croft u. a. (2011)	• Short u. a. (2007) • Kupzog (2008) • Zeilinger (2011)

Tabelle 3.1.: Eingruppierung der Arbeiten mit dem Thema Simulation von DSM

Bei den kommunikationsbasierenden Ansätzen wird eine Kommunikationsinfrastruktur eingesetzt, um die Komponenten miteinander zu verbinden und auf diesem Weg das Netz zu steuern. Ansätze aus dem Bereich des DDC nutzen keine Kommunikation zwischen den Teilnehmern um sich zu koordinieren. Vielmehr hat hier jeder Teilnehmer die Fähigkeit, die Frequenz des Netzes zu überwachen. In Abhängigkeit von der Netzfrequenz ergreift bei DDC jeder Teilnehmer selbständig vorprogrammierte Maßnahmen, um die Netzfrequenz wieder auf das normale Niveau zu bringen (Short u. a., 2007, S. 5-4).
Das in der Diplomarbeit von Kupzog (2008) beschriebene Vorgehen stellt eine Verbindung der beiden Kategorien dar. Der zur Steuerung von Netzteilnehmern entwickelte IRON-CPP-Algorithmus läuft entweder auf dem Teilnehmer selber oder auf einer an ihn angeschlossenen „DSM Interface Unit“. Zusätzlich zu dem Algorithmus, welcher als einzige Eingabegröße die aktuelle Netzfrequenz benötigt, wird die Kommunikationsinfrastruktur zur Ermittlung eines „Activation Levels“ benötigt, damit nicht eine DSM-Ressource durchgängig beansprucht wird. Vielmehr sollen, da es sich bei den bei Kupzog (2008) betrachteten Ressourcen um Energiespeicher wie z.B. Kühlschränke handelt, alle Ressourcen gleichmäßig stark belastet werden (Kupzog, 2008, S. 60-67).

Ein weiteres Kriterium zur Klassifizierung stellt die in der Arbeit schwerpunktmäßig betrachtete Netzebene dar:
Außer Gudi u. a. (2011) haben alle anderen Arbeiten eine ganzheitliche Sicht auf das Stromnetz. Gudi u. a. (2011) konzentrieren sich auf die Abläufe und das Zusammenspiel von Geräten und erneuerbaren Energiequellen innerhalb eines Haushalts. Hierzu simuliert er das Betriebsverhalten verschiedener Haushaltsgeräteklassen in Echtzeit und nutzt Partikel-Schwarm Optimierung um das DSM auf Seiten des Haushalts zu ermöglichen.

Hausnetz	Gesamtnetz
• Gudi u. a. (2011)	• Short u. a. (2007) • Kupzog (2008) • Zaidi u. a. (2010) • Croft u. a. (2011) • Zeilinger (2011)

Tabelle 3.2.: Eingruppierung der Arbeiten anhand der betrachteten Netzebene

3.3. Abgrenzung

Auch diese Arbeit befasst sich, ebenso wie alle im vorherigen Abschnitt genannten Arbeiten, mit der Simulation und der Realisierung von DSM. Besonderes Augenmerk wird auf die Umsetzung einer *Demand Response*-Mechanik, wie sie im Abschnitt 2.3.2 beschrieben wurde, gelegt.
Gegenüber den vorher genannten Arbeiten liegt der Fokus dieser Arbeit auf der Netzebene, indem von einzelnen Geräten in Haushalten und Industrieanlagen abstrahiert wird. Auf Grund der flexiblen Systemarchitektur kann diese Ebene der Simulationsumgebung bei Bedarf hinzugefügt werden.
Das zentrale Abgrenzungskriterium stellt der Einsatz einer ereignisgesteuerten Architektur unter zu Hilfenahme eines *Complex Event Processing* (CEP) dar. Bisherige Ansätze verwendeten entweder eine traditionelle *Client-Server-Architektur* oder eine *serviceorientierte Architektur*. Daher ist es das erklärte Ziel dieser Arbeit Erfahrungen mit dem Einsatz einer ereignisgesteuerten Architektur im Umfeld eines Smart Grids zu sammeln und mit ihrer Hilfe ein in der Simulation funktionierendes DSM zu entwickeln.

Im Gegensatz zu Arbeiten, die sich auf die Simulation von Smart Grids konzentrieren, liegt der Fokus bei der Entwicklung der Simulationsumgebung nicht auf einer hundertprozentig realistischen Simulation des Stromnetzes. Daher wird die Bezeichnung „Experimentierumgebung“ auch als zutreffender angesehen.
Soweit es möglich ist, sollen die verwendeten Simulationsmodelle und Verfahren möglichst einfach und anschaulich gehalten werden. So kann die entstandene Simulationsumgebung später für Demonstrationszwecke und zur Wissensvermittlung genutzt werden. Daher wird viel Wert auf die Interaktivität der Simulation gelegt, Szenarien können zur Laufzeit verändert und beeinflusst werden, was GridLAB-D und HOMER mit der Simulation von vorkonfigurierten Szenarien nicht bieten.

4. Systemarchitektur

In diesem Kapitel soll der Aufbau des Micro-Grids aus Sicht der Informatik vorgestellt werden. Zuerst wird der allgemeine Aufbau beschrieben. Anschließend wird auf die einzelnen Netzteilnehmer, wie Steuereinheiten und Kraftwerke, mit ihrer Funktionalität eingegangen.
Im zweiten Teil steht die technische Architektur und somit die Kommunikationsinfrastruktur im Fokus. Erläutert wird, wie Netzteilnehmer miteinander kommunizieren können und wie einzelne Bestandteile, wie Monitoring- und Simulationsumgebung, an die Infrastruktur angebunden sind.

4.1. Netzarchitektur

Dieser Abschnitt betrachtet die Architektur der Netzebene und damit den logischen Aufbau des Micro-Grids. Hierbei sollen die einzelnen Komponenten vorgestellt und ihr Zusammenspiel erläutert werden. Abschließend wird beschrieben, in welcher Form und durch welche Komponenten die Simulation der Netzbestandteile erfolgt.
Im weiteren Verlauf wird sich dazu an dem in Abbildung 4.1 dargestellten schematischem Aufbau des Netzes orientiert. Der Aufbau leitet sich aus dem fachlichen Datenmodell des Anwendungsbereichs ab, welches der Vollständigkeit halber im Anhang unter Abbildung A.1 hinterlegt ist.

Der Entwurf der Architektur lehnt sich an der bei Stimoniaris u. a. (2011, S. 1) beschriebenen SIM-Architektur an. Das *Smart Integration Module* (SIM) hat bei Stimoniaris u. a. (2011, S. 2) die zentrale Aufgabe das Micro-Grid zu verwalten und zu steuern. Es kann darüber entscheiden, ob das Micro-Grid im Island-Mode oder verbunden mit dem öffentlichen Stromnetz arbeiten soll. Außerdem ist es für Maßnahmen zur Lastverteilung zuständig und stellt einen Bus für den Anschluss von Gleich- und Wechselstromteilnehmern zur Verfügung.
Im Gegensatz zu Stimoniaris u. a. (2011) werden bei dieser Betrachtung die elektrotechnischen Details vernachlässigt, da sich diese Arbeit auf die Sicht der Informatik konzentriert. Es wird vielmehr davon ausgegangen, dass alle Netzkomponenten über die, in Abschnitt 4.2 genauer beschriebene, Kommunikationsinfrastruktur miteinander verbunden sind. Die Anbindung aus elektrotechnischer Sicht werden als gegeben angenommen.

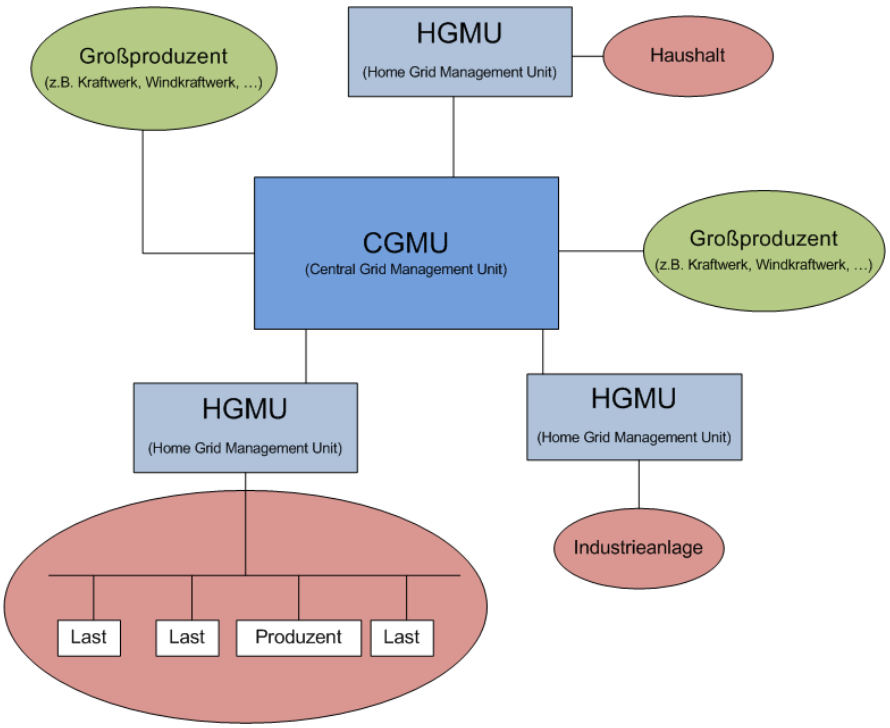

Abbildung 4.1.: Beispielhafte Darstellung des Netzaufbaus und der beteiligten informationstechnischen Komponenten

4.1.1. Central Grid Management Unit

Die *Central Grid Management Unit* (CGMU) nimmt in der Architektur den Platz der zentralen Steuerungskomponente, analog zur SIM, ein. Ihre Aufgabe besteht in der:

1. Verwaltung der angeschlossenen Netzteilnehmer.
2. Protokollierung von Stromverbrauchs-, Stromproduktions- und regelbaren Energiemengen.
3. Planung der Maßnahmen zur Netzsteuerung und damit Durchführung des eigentlichen DSM.

Um dieser Aufgabe nachgehen zu können, werden der CGMU von den angeschlossenen Netzteilnehmern die benötigten Daten, in Form von Ereignissen, über die Kommunikationsinfrastruktur mitgeteilt. Die CGMU wertet die Ereignisse aus und ermittelt einen Fahrplan zur Netzsteuerung. Dieser stellt eine Menge von Aktionen dar, welche von den Teilnehmern auszuführen sind, um angebotene und nachgefragte Strommenge ins Gleichgewicht und die Netzfrequenz auf ihren Richtwert zu bringen.

Die Ebene, also die Gesamtheit der Teilnehmer, welche die CGMU wahrnimmt, wird dabei im Folgenden als die „Netzebene“ bezeichnet. Im Einzelnen handelt es sich dabei um die Großproduzenten, also alle Formen von Kraftwerken und um die Steuereinheiten der Abnehmer, wie Haushalte und

gewerbliche Betriebe.
Die CGMU wird in dieser Arbeit schrittweise entwickelt und stellt daher eine reale Komponente dar. Dass heißt sie wird nicht simuliert und ist damit kein Teil der Simulationsumgebung.

4.1.2. Home Grid Management Unit

Bei der *Home Grid Management Unit* (HGMU) handelt es sich um die Steuereinheit für die „Hausnetzebene“. Sie wurde abweichend zur SIM-Architektur eingeführt, um eine weitere Indirektionsebene zur besseren Verteilung der Rechenlast im Netz zu schaffen.
Im Grunde sind die Aufgaben der HGMU mit denen der CGMU identisch. Sie unterscheiden sich allein in den Einheiten welche sie verwalten. Sind es bei der CGMU die großen Netzteilnehmer, steuert die HGMU die einzelnen Gerätschaften der Teilnehmer und abstrahiert so von diesen.
So gehört zu den Aufgaben der HGMU:

1. Verwaltung der Geräte eines Netzteilnehmers
2. Protokollierung von Stromverbrauchs-, Stromproduktions- und regelbaren Energiemengen aller Geräte.
3. Planung und Steuerung auf Ebene der Hausnetze.

Zusätzlich werden folgende Aufgaben in Verbindung mit der CGMU wahrgenommen:

4. Weiterleitung der regelbaren Energiemengen an die CGMU.
5. Kumulierung der Produktions- und Verbrauchswerte sowie deren Übermittlung zur CGMU.
6. Weiterleitung bzw. Umsetzung der von der CGMU erhaltenen Steuerungsanweisungen.

Die HGMU repräsentiert gegenüber der CGMU jeweils einen Haushalt oder einen Betrieb und abstrahiert von den in den Einheiten enthaltenen Maschinen und Geräten. Die CGMU muss also nicht jedes einzelne Gerät kennen, sondern erhält von der HGMU die gesammelten Verbrauchs- und Produktionsdaten. Hierdurch findet eine deutliche Reduktion der Ereignismenge und des Zustandsraums statt.
Gleichzeitig teilt sie ihr mit, welche Steuerungsmöglichkeiten, also z.B. welche Verbraucher an- oder abgeschaltet werden können, überhaupt vorhanden sind, und sogt nach dem Erhalt der entsprechenden Anweisungen für deren Umsetzung.

Simulation

Eine Implementierung der HGMU im Rahmen dieser Arbeit wird nicht angestrebt. Vielmehr wird von den einzelnen Geräten und Maschinen abstrahiert, indem die Haushalte und Industriebetriebe durch den Einsatz von Lastkurven simuliert werden.
Stromversorgungsunternehmen nutzen bei Netzteilnehmern mit einem Jahresverbrauch von unter

100 000 kW h synthetische Lastprofile zur Vorhersage der Verbrauchswerte (E.ON Mitte AG, 2013, 1). Neben dem Einsatz als Mittel zur Verbrauchsvorhersage werden, u.a. bei Kupzog (2008, S. 13), Lastprofile zur Simulation der Hausnetzebene eingesetzt. Im Rahmen dieser Arbeit werden hierzu die Lastprofile *H0* und *G0* genutzt. Das Profil *H0* wird für Haushalte und solche mit geringfügig gewerblichen Bedarf eingesetzt. *G0* steht für allgemeines Gewerbe und stellt den Durchschnitt über die Profile *G1* bis *G6* dar. Bei beiden Profilen wird jeweils das Verbrauchsverhalten für einen typischen Werktag der Übergangsperiode zur Simulation des Verhaltens der Netzteilnehmer verwendet. Das verwendete Profil ist im Anhang in Abbildung B.1 abgebildet.
Die so entstandenen Simulationsmodelle werden um die Fähigkeit zur Ereignisverarbeitung erweitert, so dass keine explizite Simulation der HGMU nötig ist. Vielmehr findet sich ihre Funktionalität in den Simulationsmodellen wieder.

4.1.3. Großproduzenten

Die restlichen Großproduzenten, wie Windkraftanlagen und konventionelle Kraftwerke, kommunizieren direkt mit der CGMU. Bei einer späteren Umsetzung könnten auch hier HGMU genutzt werden, um z.B. mehrere Blockheizkraftwerke oder Windräder zu größeren Einheiten (*virtuellen Kraftwerken*) zusammenzuschließen.
Während der Simulation können einzelne Produzenten dem Netz hinzugefügt werden, die direkt mit der CGMU kommunizieren. Daher wurde hier auch auf den Einsatz vom HGMUs verzichtet und die Funktionalität zur Ereignisverarbeitung in die Simulationsmodelle übernommen.

4.2. Technische Infrastruktur

Das Herz der ereignisgesteuerten Architektur der Simulationsumgebung und des Micro-Grids bildet eine *Message Oriented Middleware*. Über diese können die Komponenten mit den Steuereinheiten kommunizieren und tauschen so die Ereignisse miteinander aus.

Als Kommunikationsprotokoll wurde sich, wie in Abbildung 4.2 dargestellt, für *MQTT*[1] entschieden. Ursprünglich durch IBM entwickelt, handelt es sich bei *MQTT* um ein leichtgewichtiges Nachrichtenprotokoll, welches nach dem Publish-Subscribe-Prinzip arbeitet. Auf Grund seiner geringen Anforderungen an Bandbreite und der direkten Unterstützung von QoS-Schemata, wird *MQTT* besonders im Bereich der Machine-To-Machine-Kommunikation (M2M) und dem Internet der Dinge eingesetzt. Dank dieser Eigenschaften und dem Vorhandensein von Implementierungen auch für eingebettete Systeme, wird *MQTT* als eine ideale Basis zur Realisierung der ereignisgesteuerten Architektur angesehen. Als *Message Broker* wird *ActiveMQ*[2] verwendet, der neben anderen Protokollen auch *MQTT* unterstützt.

[1] `http://www.mqtt.org`
[2] `http://activemq.apache.org/`

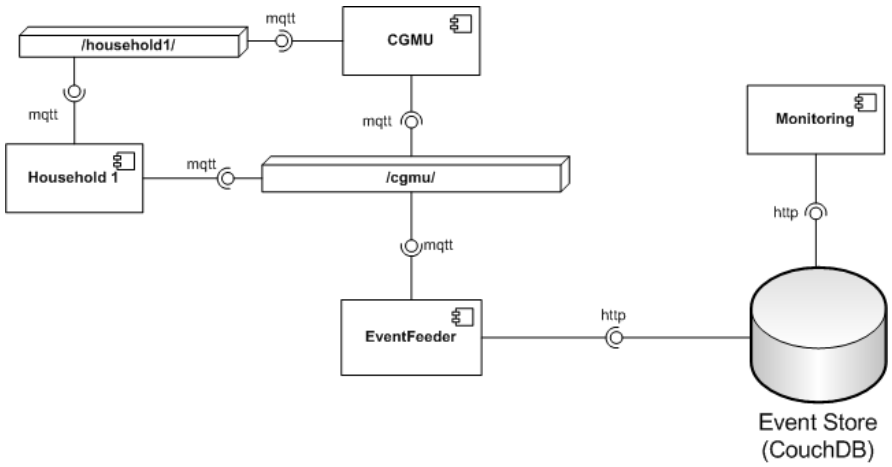

Abbildung 4.2.: Technische Architektur des Micro-Grids

4.2.1. Kommunikation

Damit die einzelnen Komponenten einander finden und miteinander kommunizieren können, unterschreiben sie zunächst für ein Topic mit ihrer ID als Bezeichner. Also trägt sich die CGMU für Nachrichten von dem Topic „/cgmu/“ ein. Dadurch, dass jeder Teilnehmer unter seiner ID erreichbar ist, ist das „Antworten“ auf Ereignisse einfach, da das Topic, durch die in der Nachricht übermittelte ID, bekannt ist.
Für diese Arbeit wurde der Aspekt der Sicherheit bzw. Absicherung der Kommunikationsinfrastruktur und der Topics erst einmal außen vorgelassen. Für ein tragfähiges Gesamtkonzept ist dieser Aspekt in weiteren Arbeiten genauer zu untersuchen, bevor ein hier beschriebenes System in den Produktivbetrieb gehen könnte.

4.2.2. Anbindung der Simulationsumgebung

Die Simulationsumgebung beinhaltet die zu simulierenden Netzteilnehmer, also z.B. Haushalte, Windkraftwerke usw.. In Abbildung 4.2 ist sie nicht explizit dargestellt und wird durch die Komponente „Household1“ repräsentiert.
Da jedes Simulationsmodell eines Netzteilnehmers Ereignisse mit der CGMU austauschen muss, besitzen sie alle eine Abhängigkeit zur Kommunikationsinfrastruktur. Um diesen schwergewichtigen Service in der Simulationumgebung zentral bereitzustellen, wurde der in Abbildung 4.3 dargestellte Ansatz, eines zentralen Kommunikationsadapters gewählt.
Alle Simulationsmodelle, die vereinfacht aus dem eigentlichen Modell und einem Kommunikationsmodul zur Ereignisverarbeitung und -versand bestehen, greifen über die Schnittstellen *IMessagePublishingService* und *IMessageNotificationService* auf den MessagingAdapter zu. Dieser kapselt

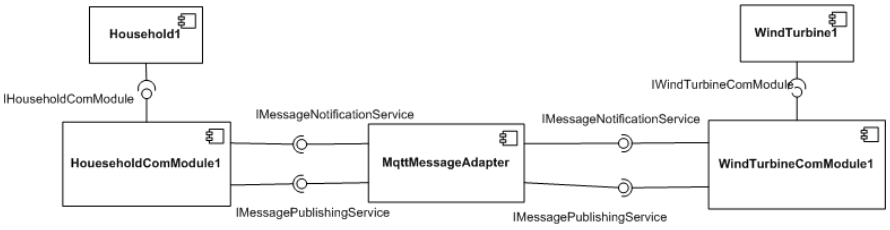

Abbildung 4.3.: Anbindung der Simulationsumgebung an die Kommunikationsinfrastruktur

den konkreten Zugriff auf die Kommunikationsinfrastruktur. Beim Entwurf wurde sich an der bei Siedersleben (2004) beschriebenen Quasar-Softwarearchitektur orientiert.

4.2.3. Monitoring

Für die Überwachung und Protokollierung der Simulation wird ein Werkzeug benötigt, welches die an die CGMU gesandten Ereignisse protokolliert und persistiert. Die CGMU ist die einzige Komponente im System welche den Gesamtzustand des Systems kennt, so dass hier eine Monitoring Komponente ansetzen muss. Da die Simulation in nahezu Echtzeit arbeitet, muss auch das Monitoring diese Anforderung erfüllen.
Da weder MQTT noch ActiveMQ die Möglichkeit besitzen Kommunikationsabläufe zu protokollieren und da im Sinne des Separation-of-Concerns-Prinzips davon Abstand genommen wurde das Monitoring als Teil der CGMU zu implementieren, wurden mehrere unabhängige Komponenten entwickelt welche die Funktionalität des Monitoring implementieren. Diese sind in Abbildung 4.2 auf der rechten Seite dargestellt und sollen hier kurz vorgestellt und ihr Zusammenwirken erläutert werden.

Der *EventFeeder* stellt das Bindeglied zwischen Monitoring und der Kommunikationsinfrastruktur dar. Er hört auf das Topic der CGMU und greift die Ereignisse dort ab. Danach werden die Ereignisse für eine dauerhafte Speicherung in die NoSQL-Datenbank *CouchDb*[3] geschrieben.
Durch die Speicherung der Ereignisse kann bei Bedarf der Historische Verlauf eines Simulationslaufs nachvollzogen werden. Die eigentliche Darstellung des Simulationszustandes, dass heißt Produktions-, Verbrauchswerte und aktuelle Netzfrequenz, wird von einer Webanwendung übernommen. Sie stellt damit die eigentliche Monitoringkomponente dar.
Um immer über die neusten Ereignisse informiert zu sein, wird in der Webanwendung der Benachrichtigungsmechanismus[4] von *CouchDB* genutzt. Dieser ist auch ereignisgesteuert und somit wird auch hier der architektonische Ansatz durchgängig verfolgt. Die Kommunikation zwischen *EventFeeder*, *CouchDB* sowie *Monitoringanwendung* erfolgt über HTTP. *CouchDB* stellt für den Zugriff eine REST-basierte API zur Verfügung, die von den beiden zugreifenden Komponenten genutzt wird.

[3] `http://couchdb.apache.org/`
[4] `http://docs.couchdb.org/en/latest/changes.html`

5. Analyse

Ausgehend von der im vorangegangenen Kapitel ermittelten Netzarchitektur, sollen an dieser Stelle die identifizierten Teilsysteme genauer betrachtet werden. Hierbei wird sich zunächst in diesem Kapitel auf die Analyse der an die Teilsysteme gestellten Anforderungen konzentriert. Die dabei ermittelten Anforderungen werden in den folgenden Abschnitten, nach Teilsystemen unterteilt, vorgestellt.
Aufbauend auf den Ergebnissen dieser Analyse, erfolgt anschließend im Kapitel 6 der architektonische Entwurf der Teilsysteme.

5.1. Beschreibung der Arbeitsweise des Systems

Zur Ermittlung der Systemanforderungen wurden zunächst die durch das System zu unterstützenden Vorgänge betrachtet. Diese sollen im Verlauf dieses Abschnittes aus Sicht der beteiligten Benutzer in Kürze beschrieben werden.

5.1.1. Schüler- und Studentensicht

Schüler und Studenten nutzen die Simulation, um sich mit Hilfe von Experimenten die komplexen Zusammenhänge und Abläufe in intelligenten Stromnetzen zu veranschaulichen. Sie verändern hierzu interaktiv die Simulation, indem sie vordefinierte Netzteilnehmer hinzufügen, entfernen oder steuern. Außerdem können sie Umweltfaktoren, wie z.B. Sonneneinstrahlung und Windgeschwindigkeit, ändern und so die Simulation beeinflussen.
Die Auswirkungen, die die Veränderungen auf das Netz haben, werden durch das System visuell veranschaulicht. Anhand dieser Eindrücke erfahren die Schüler und Studenten Näheres über die Funktionsweise des zugrundeliegenden Systems und wie sich Veränderungen auf das Stromnetz auswirken.

5.1.2. Entwicklersicht

Ein Softwareentwickler möchte die Simulation dazu nutzen um neue Ideen zu testen. Er entwickelt neue Komponenten, die in das System integriert und anschließend getestet werden. Hierdurch können die mit den Komponenten verbundenen Hypothesen im Vorfeld überprüft werden.
Um die für aussagekräftige Tests benötigten Szenarien realisieren zu können, muss es ihm möglich, sein die Simulation um neue Modelle zu erweitern und existierende zu verändern. So kann auf eine

existierende und getestete Infrastruktur aufgesetzt und zusätzlicher Aufwand für Realisierung und Tests verringert oder sogar vollständig vermieden werden. Der Entwickler kann sich voll und ganz auf die zu realisierende Fragestellung konzentrieren.

5.1.3. Autorensicht

Aus Sicht des Autors dient das System zur Überprüfung und Veranschaulichung der These, dass sich eine ereignisgesteuerte Architektur zur Realisierung eines sich autark steuernden Stromnetzes eignet.
Auf dem Weg der Planung und Realisierung sollen die dabei auftretenden Problem- und Fragestellungen untersucht werden. Anhand einer prototypischen Implementierung des Systems können anschließend die mit dem gewählten Ansatz verbundenen Vor- und Nachteile evaluiert werden. Die bei den Testläufen gewonnen Daten sollen zum Vergleich mit anderen Ansätzen herangezogen werden.

5.2. Funktionale Anforderungen

Ausgehend von den beschriebenen Szenarien werden in diesem Abschnitt die an das System gestellten funktionalen Anforderungen ermittelt. Sie werden nach Teilsystemen getrennt beschrieben. Der folgende Abschnitt befasst sich dann mit den nichtfunktionalen Anforderungen.

5.2.1. Simulationsumgebung

Die Simulationsumgebung wird zur Gestaltung von Szenarien und der Durchführung von Experimenten genutzt. Hierzu bietet sie den Benutzern vorgefertigte Modelle und Werkzeuge zu deren Steuerung an. Außerdem stellt sie Schnittstellen und Strukturen zur Erweiterung ihrer Funktionalität zur Verfügung.
Die an sie gestellten Anforderungen wurden von den Szenarien aus Sicht der Studenten, Schüler und Entwickler abgeleitet:

SF-1) Die Umweltfaktoren Sonneneinstrahlung und Windstärke sollen durch die Umgebung simuliert werden.

SF-2) Die Simulationszeit ist ein weiterer Umweltfaktor.

SF-3) Die Umweltfaktoren sollen sich während der Simulation durch die Benutzer verändern lassen.

SF-4) Netzteilnehmer können während der Simulation hinzugefügt werden.

SF-5) Netzteilnehmer können während der Simulation gesteuert werden.

SF-6) Netzteilnehmer können während der Simulation aus dem Netz entfernt werden.

SF-7) Die Simulationsumgebung stellt Modelle der folgenden Netzteilnehmer bereit:

a) Windkraftanlage

b) Solarkraftwerk

c) Energiespeicher

d) Haushalte

e) Industrieanlagen

SF-8) Die Umgebung bietet die Möglichkeit zur Erweiterung mit neuen Modelle und Funktionen an.

SF-9) Das Verhalten bestehender Modelle soll sich anpassen lassen.

SF-10) Die Simulationsmodelle teilen dem System ihre Zustandsänderungen über Ereignisse mit.

SF-11) Aktionen zur Änderung eines Modellzustandes können über Ereignisse durch das System veranlasst werden.

5.2.2. Monitoring

Das zweite Teilsystem, das Monitoring, dient zur Überwachung und Veranschaulichung des Stromnetzzustandes. Es wertet die im System versandten Ereignisse aus, extrahiert die für den Systemzustand wichtigen Daten und stellt den Systemzustand grafisch dar.
Es wird von allen beschriebenen Nutzern gleichermaßen verwendet. Schüler und Studenten können sich anhand der Daten die Auswirkungen von Systemeingriffen verdeutlichen. Entwickler können anhand der Daten die Funktionsweise der zu testenden Komponenten verifizieren. Und im Rahmen dieser Arbeit können auf diesem Weg die Simulationsdaten erhoben, sowie die Funktionsweise der Netzsteuerung überprüft werden.
Es wurden die folgenden Anforderungen aus den drei beschriebenen Sichten abgeleitet:

MF-1) Es sollen die aktuellen Werte und der Verlauf von produzierter und konsumierter elektrischer Energie protokolliert werden.

MF-2) Energieproduktion und -verbrauch sind grafisch darzustellen.

MF-3) Die aktuelle Netzfrequenz und ihr Verlauf sind zu erfassen.

MF-4) Der Verlauf der Netzfrequenz ist grafisch darzustellen.

MF-5) Die am System angemeldeten Netzteilnehmer sind zu erfassen.

Zusätzlich zu den beschriebenen Szenarien bauen die hier formulierten Anforderungen auf den physikalischen Grundlagen von Stromnetzen auf. Hierbei wurden ausgehend von den in Abschnitt 2.1.2 beschriebenen Zusammenhängen zwischen Energieproduktion, -verbrauch und Netzfrequenz,

diese drei Größen als zentrale Merkmale für den Netzzustand identifiziert.
Ihre Überwachung ermöglicht es zu überprüfen, ob ergriffene Maßnahmen den gewünschten Effekt auf den Netzzustand haben und ob die eingesetzte Steuerung einen stabilen Betrieb des Stromnetzes gewährleisten kann.

5.2.3. Central Grid Management Unit

Das letzte Teilsystem ist die Central Grid Managment Unit (CGMU). Wie im Abschnitt 4.1 beschrieben, handelt es sich bei ihr um die zentrale Steuerungseinheit des Netzes. Ihre Funktionalität definiert die Art, wie das Netz gesteuert wird und ist damit der zentrale Gegenstand der Untersuchungen in dieser Arbeit. Die an sie gestellten Anforderungen sind insbesondere im Kontext der ereignisgesteuerten Architektur und der Steuerung des Netzes zu sehen:

CF-1) Verarbeitung von Beitritten von Netzteilnehmern zum Netz.

CF-2) Verarbeitung von Austritten aus dem Netz.

CF-3) Überwachung von Änderungen der produzierten Energiemenge im Netz.

CF-4) Überwachung des Energieverbrauchs im Stromnetz.

CF-5) Überwachung der Netzfrequenz.

CF-6) Protokollierung von durch Netzteilnehmer angebotenen Steuerungsmaßnahmen.

CF-7) Planung von Maßnahmen zur Steuerung des Stromnetzes.

CF-8) Ergreifen von Steuerungsmaßnahmen bei einer Abweichung von der Normfrequenz größer als $\pm 0{,}02\,\mathrm{Hz}$.

CF-9) Durchführung der ermittelten Steuerungsmaßnahmen veranlassen.

Auch bei diesem Teilsystem spielt der Netzzustand eine wichtige Rolle. Abhängig von der Netzfrequenz muss die CGMU Maßnahmen, ergreifen um das System in einen stabilen Zustand zu überführen.
Daher kommen die beschrieben Anforderungen aus den Bereichen Überwachung des Netzzustandes und der Planung von Steuerungsmaßnahmen. Da hierbei durchaus unterschiedliche Ansätze und Strategien verfolgt werden können, sind hier weitere nichtfunktionale Anforderungen zu berücksichtigen. Diese werden im Abschnitt 5.4 näher erläutert.

5.3. Übergeordnete Anforderungen

Bei der Analyse des Gesamtsystems konnten Anforderungen ermittelt werden, die an alle Teilsysteme gestellt werden. Die teilsystemübergreifenden Anforderungen gehen auf die benötigte Interoperabilität zwischen den Systemen zurück und sind durch die zugrundeliegende ereignisgesteuerte

Architektur geprägt.
Damit sich die Teilsysteme untereinander verständigen können, müssen sie die Fähigkeit zur Verarbeitung der fachlichen Ereignisse des Anwendungsbereichs besitzen. Hierzu müssen sich die Teilsysteme vorher darüber verständigt haben, welche Ereignisse aus dem Anwendungsbereich heraus benötigt werden. Dieses geschieht meist durch ein zentrales Ereignismodell.
Die folgenden Anforderungen sind allen Teilsystemen gemein:

ÜF1) Einsatz fachlicher Ereignisse zur Kommunikation zwischen Teilsystemen.

ÜF2) Fähigkeit zur Verarbeitung einer Teilmenge der im Ereignismodell beschriebenen fachlichen Ereignisse.

Zusätzlich werden an die Art der Ereignisverarbeitung noch Anforderungen aus dem Bereich der „Änderbarkeit“ nach ISO/IEC 9126 gestellt. Da diese aber zu dem Bereich der nichtfunktionalen Anforderungen zählen, sei an dieser Stelle auf den nächsten Abschnitt verwiesen, in dem diese Art von Anforderungen genauer betrachtet werden.

5.4. Nichtfunktionale Anforderungen

Im Gegensatz zu den funktionalen Anforderungen, die beschreiben was ein System leisten soll, beschrieben die nichtfunktionalen Anforderungen die Güte der Leistungserbringung. Die strukturierte Bestimmung der nichtfunktionalen Anforderungen orientiert sich an den in der Norm *ISO/IEC 9126* beschriebenen Kategorien. Diese sollen in den kommenden Abschnitten einzeln betrachtet und so die für die Arbeit relevanten Anforderungen extrahiert werden.

5.4.1. Zuverlässigkeit

Der Bereich der Zuverlässigkeit ist für die Stromversorgung von immenser Bedeutung. Eine stabile und zuverlässige Stromversorgung stellt einen wichtigen Wirtschaftsfaktor und mittlerweile sogar die Grundlage unseres täglichen Lebens dar.
Stromausfälle führen nicht nur zu großen wirtschaftlichen Schäden, sondern können, sobald sie über einen längeren Zeitraum andauern, zu kritischen Versorgungssituationen führen (vgl. Mensk und Gardemann (2008); Bundesamt für Bevölkerungsschutz und Katastrophenhilfe (2011)). Daher werden an Stromnetze grundsätzlich besonders hohe Anforderungen in Bezug auf die Versorgungsstabilität und -güte gestellt.
In dieser Arbeit soll es jedoch vornehmlich, um die Evaluierung von Entwurfsmöglichkeiten und eine prototypische Umsetzung der Netzsteuerung gehen. Daher wird das in der Realität so außerordentlich wichtige Kriterium der Zuverlässigkeit hier vernachlässigt.

Im Rahmen eines Praxiseinsatzes würden in den Teilbereichen *Fehlertoleranz*, *Wiederherstellbarkeit* und insbesondere bei der *Konformität* hohe Anforderungen an das System gestellt.

Im Bereich der *Fehlertoleranz* soll die lose Kopplung der Komponenten und die verwendete Architektur dazu beitragen, dass der Ausfall einzelner Systemkomponenten nicht zum Ausfall des Gesamtsystems führt. Da der Anwendungsbereich jederzeit vorsieht, dass Netzteilnehmer dem Netz beitreten und es wieder verlassen können, ist hierdurch schon ein gewisser Teilbereich des Funktionsumfangs abgedeckt. Trotzdem werden folgende Anforderungen an das System formuliert:

NF1) Der Ausfall einzelner Komponenten darf nicht den Ausfall des Gesamtsystems zur Folge haben.

NF2) Die CGMU ist so auszulegen, dass sie selbständig den Ausfall von Netzteilnehmern erkennt und darauf reagiert.

NF3) Da die CGMU und die zum Einsatz kommende Middleware die zentralen Komponenten des Systems darstellen, sind diese redundant und ausfallsicher auszulegen, um einen *Single Point of Failure* im System zu vermeiden.

Der Aspekt der *Wiederherstellbarkeit* ist genau so von enormer Bedeutung für den Praxiseinsatz. Besonders die Fähigkeit der „Selbstheilung“ wird bei der Betrachtung von Smart Grids und Micro-Grids hervorgehoben. Das Netz soll dabei in der Lage sein, nach dem Ausfall von Komponenten nicht nur weiterzuarbeiten, sondern soll auch selbständig benötigte Komponenten wiederherstellen können.
Diese Fähigkeit wird erst in einer späteren Ausbaustufe des Systems angestrebt und liegt daher in dieser Arbeit noch nicht im Fokus. Anforderungen an das System aus diesem Bereich werden daher nicht erhoben.
Jedoch sind geeignete Ansätzen ersichtlich, um dem System diese Fähigkeit zu ermöglichen. Im Bereich der Telekommunikationsinfrastruktur, ebenso eine zentrale Infrastruktur wie das Stromnetz mit sehr ähnlichen Anforderungen, hat sich das Aktorenmodell bewährt (Hewitt u. a., 1973; Hewitt, 2010). Im Telekommunikationsbereich hat sich dabei besonders die Implementierung in Erlangs *Open Telecom Platform* (Erlang OTP) etabliert, die statt der schwierigen Vermeidung die Behandlung von Fehlern in den Vordergrund stellt (Armstrong, 2007, S. 329 ff.). Die Verwendung eines eben solchen Ansatzes scheint auch für die Umsetzung von Anforderungen aus dem Bereich der *Wiederherstellung* vielversprechend zu sein.

5.4.2. Benutzbarkeit

Anforderungen zur Benutzbarkeit ergeben sich aus der Bedienung der Simulationsumgebung, da hier die eigentliche Interaktion der Endbenutzer mit dem System erfolgt. Da das Steuerungssystem nach seiner Fertigstellung seine Arbeit selbstständig verrichten soll, findet ein direkter Zugriff nur zu den Zwecken Störungsbeseitigung, Wartung und Überwachung durch Fachpersonal statt. Auch in dieser Zielgruppe werden natürlich Anforderungen an die Benutzbarkeit gestellt. Da es sich dabei aber um die Phase des Systembetriebs handelt, die nicht im Fokus der Arbeit liegt, wird sich in diesem Abschnitt auf die Anforderungen zur Nutzung der Simulationsumgebung und nicht der

Netzsteuerung konzentriert.

Damit die Schüler und Studenten ohne größeren Einarbeitungsaufwand die Simulationsumgebung bedienen können, ist besonders auf eine intuitive Bedienbarkeit wert zu legen. Um diese erreichen zu können, sind die folgenden Anforderungen zu erfüllen:

NF4) Das Bedienkonzept soll gängigen Mustern zur Interaktion mit der Benutzeroberfläche folgen.

NF5) Bei der Entwicklung des Bedienkonzeptes ist die Möglichkeit zur späteren Portierung der Simulationsumgebung auf einen *Microsoft PixelSense*[1] zu berücksichtigen.

NF6) Es soll ein visuell geprägtes Bedienkonzept entwickelt werden, daher wird die Verwendung von Piktogrammen und Symbolen angestrebt.

NF7) Änderungen des Netzzustandes sollen direkt in der Simulationsumgebung für die Benutzer ersichtlich sein.

5.4.3. Effizienz

Anzustreben ist weiterhin ein effizienter Umgang mit den Ressourcen des Stromnetzes. Dieser wird als allgegenwärtiges Entwurfsziel angesehen und daher nicht gesondert betrachtet. Das Augenmerk liegt auf der Netzsteuerung und somit auf den Größen Energieproduktion und -verbrauch. Es soll das Ziel sein, eine möglichst umweltschonende Regelung des Stromnetzes zu erreichen. Hierzu sind die Mengen an produzierter und verbrauchter Energie zu minimieren. Hieraus ergeben sich folgende Anforderungen:

NF8) Die Netzsteuerung soll eine möglichst umweltschonende Regelung des Stromnetzes umsetzen.

NF9) Die Menge an verbrauchter Energie soll minimiert werden.

NF10) Die Menge produzierter Energie, insbesondere aus konventionellen Energiequellen, ist zu minimieren.

5.4.4. Änderbarkeit

Im Abschnitt 5.2 wurde bereits auf die funktionalen Anforderungen bezüglich der Erweiterbarkeit der Simulation und das Hinzufügen sowie Testen weiterer Teilsysteme eingegangen. Aus Sicht der nichtfunktionalen Anforderungen liegt der Schwerpunkt auf der *Testbarkeit* und der Dokumentation. Diese unterstützen die funktionalen Anforderungen, die an das System bezüglich seiner Erweiterbarkeit gestellt werden.
Ein konsequentes testgetriebenes Vorgehen ermöglicht es Fehler, die bei der Änderung bestehender Funktionen oder bei der Erweiterung des Systems um neue Komponenten passieren können, einfach

[1] `http://www.microsoft.com/en-us/pixelsense/`

entdecken und nachvollziehen zu können. Eine ausführliche Dokumentation der Erweiterungspunkte unterstützt dies zusätzlich.

NF11) Es soll eine tesgetriebene Entwicklung verfolgt werden.

NF12) Erweiterungspunkte des Systems sollen ausführlich dokumentiert werden.

5.4.5. Übertragbarkeit

Hier ergeben sich aufgrund der zugrundeliegenden Architektur keine zusätzlichen nichtfunktionalen Anforderungen. Die Komponenten lassen sich austauschen und übertragen, solange sie das fachliche Ereignismodell unterstützen und mit der eingesetzten Middleware kommunizieren können. Dadurch können die Teilsysteme auch auf anderen Knoten installiert oder dorthin übertragen werden, solange sie die gerade genannten Bedingungen weiterhin erfüllen. Somit konnten in diesem Anforderungsbereich keine zusätzlichen nichtfunktionalen Anforderungen ausgemacht und definiert werden.

5.5. Zusammenfassung

In diesem Kapitel wurden zuerst die verschiedenen Nutzersichten auf das System beschrieben. Sie stellen die unterschiedlichen Erwartungshaltungen der Nutzer an das Arbeiten mit dem System dar. Während Schüler und Studenten die Simulation zum Experimentieren und zum Aufbau von Wissen nutzen sollen, sind Entwickler an der Nutzung des Systems zum Testen und Verifizieren von neu entwickelten Komponenten interessiert. Im Rahmen dieser Arbeit dient die prototypische Umsetzung einer autonomen Steuerung für Micro-Grids der Prüfung, ob Ereignisse und eine ereignisgesteuerte Architektur hierzu ein probates Mittel sind.
Aufbauend auf den darin beschriebenen Szenarien wurden in Abschnitt 5.2 zunächst die funktionalen Anforderungen an das System bzw. die einzelnen Teilsysteme ermittelt. Im Anschluss wurden die nichtfunktionalen Anforderungen in Abschnitt 5.4 ermittelt. Dabei anhand der durch die Norm ISO/IEC 9126 spezifizierten Teilbereichen nach Anforderungen gesucht, welche bei der Entwicklung des Systems berücksichtigt werden müssen.
Auf Basis der gewonnen Anforderungen und Erkenntnisse erfolgt in Kapitel 6 der Komponentenentwurf der einzelnen Teilsysteme, die im Anschluss zu der Systemarchitektur verbunden werden.

6. Softwarearchitektur

In diesem Kapitel sollen die Architekturen der zentralen Systemkomponenten vorgestellt und erläutert werden. Hierbei handelt es sich im Einzelnen um die Architekturen der Simulationsumgebung, des Monitorings und der Central Grid Management Unit. Die Grundlage aller Architekturen bildet die, in Abschnitt 4.2 beschriebene, technische Infrastruktur, welche die Teilsysteme untereinander verbindet. Hierfür wird auf den im vorangegangenem Kapitel formulierte funktionalen und nichtfunktionalen Anforderungen aufgebaut.

6.1. Architektur der Simulationsumgebung

Der zentraler Aspekt bei dem Entwurf der Architektur der Simulationsumgebung bestand in der Entwicklung einer modularen Komponentenstruktur für Netzteilnehmer, welche die in Abschnitt 5.2 ermittelten funktionalen sowie nichtfunktionalen Anforderungen, bezogen auf die Veränderbarkeit und die Erweiterbarkeit, erfüllt.
Ausgangspunkt der Überlegungen war die Orientierung an einem Modulbaukastensystem, wie es in der Industrie, insbesondere in der Autoindustrie, etabliert ist. Jedes Modul (Komponente) besitzt seinen dedizierten Aufgabenbereich und arbeitet mit anderen Modulen über standardisierte Schnittstellen zusammen. Im Verbund realisieren sie die Gesamtfunktionalität des Netzteilnehmers.
Die daraus resultierende Architektur erlaubt eine einfache Identifizierung von Ansatzpunkten zur Erweiterung und Änderung. Zusätzlich kann sie als Checkliste, zur Strukturierung des Vorgehens bei Entwicklung neuer Netzteilnehmer, genutzt werden.

Abbildung 6.1 zeigt die allen Netzteilnehmern zugrundeliegende Komponentenarchitektur. Bei den rot umrandeten Komponenten handelt es sich um die eigentlichen Komponenten der Simulationsumgebung, während die restlichen Komponenten die Basisarchitektur eines Netzteilnehmers darstellen.
Die Simulationsumgebung bietet Fabrikkomponenten zum Bau der Netzteilnehmer an. Sie kapseln die Auflösung der Abhängigkeiten zwischen den Komponenten der Netzteilnehmermodelle und stellen den Zugriff auf die simulationseigenen Komponenten wie *Environment* und *CommunicationAdapter* zur Verfügung.
Die Komponente *Environment* ist dabei für die Simulation und Steuerung der Umweltfaktoren zuständig. Während der Kommunikationsadapter die Verbindung zur eingesetzten Middleware herstellt und damit die Kommunikation mit anderen Teilsystemen ermöglicht.
Beide Komponenten sollen im Anschluss kurz vorgestellt und beschrieben werden. Danach wird die

Architektur der Netzteilnehmer noch einmal genauer betrachtet.

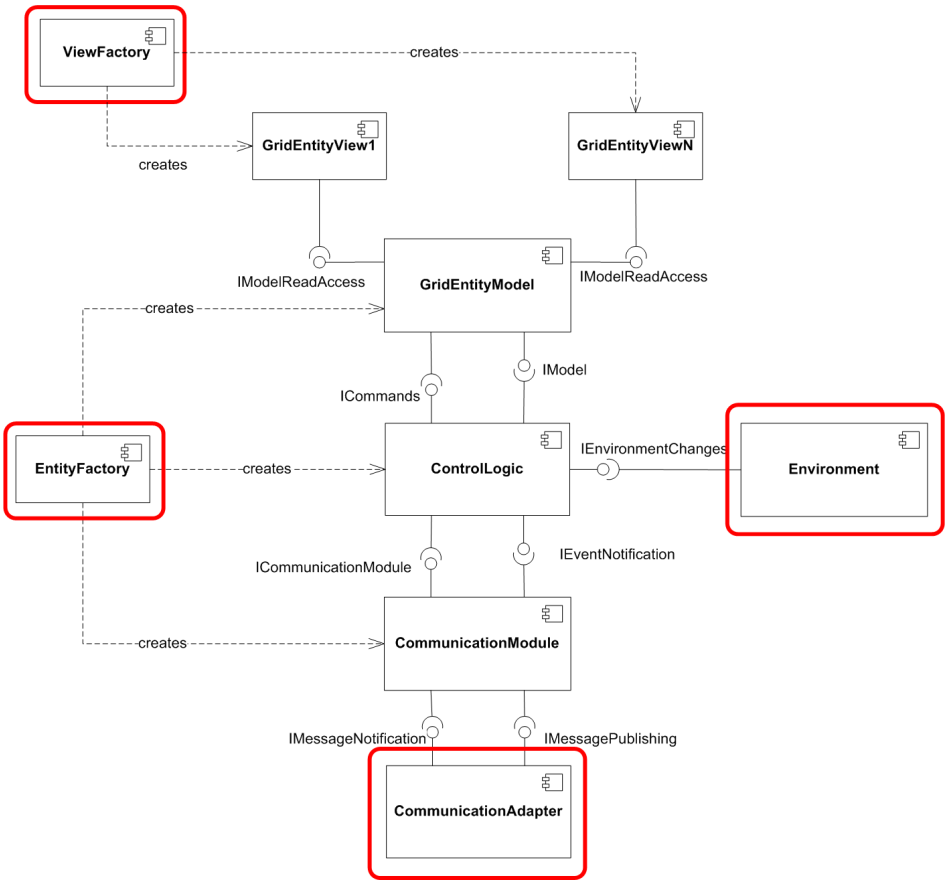

Abbildung 6.1.: Architektur der Simulationsumgebung anhand eines Beispielsimulationsmodells

6.1.1. Environment

Die Komponente Environment verwaltet und simuliert die Umweltfaktoren der Simulation. Außerdem benachrichtigt sie die Simulationsmodelle, wenn sich die für das jeweilige Modell relevanten Bedingungen geändert haben. Während der Analyse wurde ermittelt, dass die folgenden Umweltfaktoren für die Durchführung von Simulationen benötigt werden:

- Windgeschwindigkeit $[\mathrm{km/h}]$
- Sonneneinstrahlung $[\mathrm{W/m^2}]$
- Datum und Uhrzeit

Bei der Windgeschwindigkeit und der Sonneneinstrahlung handelt es sich um (Double-)Werte, welche zur Laufzeit in einem bestimmten Rahmen verändert werden können. Um die Simulationszeit von der Systemzeit getrennt steuern zu können, muss die Systemuhr auf Sekundenbasis nachgebildet

werden.
Architektonisch unterscheidet sich das Umweltmodell durch die fehlende Anbindung an die Kommunikationsinfrastruktur von den anderen Netzteilnehmermodellen. Diese ist, da das Umweltmodell die Modelle der Netzteilnehmer direkt benachrichtigt, auch nicht nötig.

6.1.2. CommunicationAdapter

Der Kommunikationsadapter bietet den Netzteilnehmermodellen die Möglichkeit zum Versenden und Empfangen von Nachrichten an. Er arbeitet eng mit den Kommunikationsmodulen der Modelle zusammen und ermöglicht es, dass bei einem Wechsel der konkreten Middleware nicht alle Kommunikationsmodule angepasst werden müssen. Vielmehr wird in diesem Fall die konkrete Implementierung des Kommunikationsadapters gegen eine neue ausgetauscht. Auf Grund der durch den Adapter angebotenen Schnittstellen entstehen so für die Kommunikationsmodule keine Änderungen.

6.1.3. Netzteilnehmer

Wie eingangs angesprochen war die Idee hinter der hier beschriebenen Architektur eine einheitliche Struktur für alle Modelle zu entwickeln. Hierdurch lassen sich folgende Vorteile erreichen:

+ Einfache Erstellung und Integration neuer Modelle durch die Vorgabe einer einheitlichen Struktur.
+ Einfache Veränderbarkeit von Modelleigenschaften durch eindeutige Zuständigkeiten.
+ Das Prinzip der Ereignissteuerung findet sich nach der Realisierung auch auf der Ebene der Simulationsmodelle wieder.

Aus der Überlegung, aus welchen Modulen ein Netzteilnehmer letztendlich physisch zusammengesetzt werden würde, entstand die in Abbildung 6.1 dargestellte Architektur.
Die Hauptkomponenten sind dabei:

- Das Netzteilnehmermodell (*GridEntityModel*)
- Die Steuerungslogik (*ControlLogic*)
- Das Kommunikationsmodul (*CommunicationModule*)

Bei den unterschiedlichen Sichten auf das Modell (*GridEntityViews*) handelt es sich um softwaretechnisch motivierte Komponenten, die hier nicht näher betrachtet werden sollen.
Vielmehr erfolgt zum Abschluss dieses Abschnitts eine genauere Beschreibung der Hauptkomponenten.

Netzteilnehmermodell

Das Netzteilnehmermodell verwaltet die eigentlichen Daten des Modells und damit seinen Zustand. Dieses können z.B. aktuelle Produktions- und Verbrauchswerte sein, aber auch Hilfsfunktionen zu deren Berechnung und das Anbieten einer Schnittstelle zur Abfrage der Werte gehören zu seinen Aufgaben. Letztendlich benachrichtigt das Modell auch angeschlossene View-Objekte darüber, dass sich Werte geändert haben.

Steuerungslogik

Als Bindeglied zwischen Modell und Kommunikationsmodul fungiert die Steuerungslogik. Sie steuert welche Zustände ein Netzteilnehmer einnehmen kann. Kommen Ereignisse zur Steuerung des Teilnehmers am Kommunikationsmodul an, reagiert die Steuerungslogik darauf und ändert ggf. den Zustand des Teilnehmers. Weiterhin sorgt sie dafür, dass die Werte des Modells immer mit denen für den Zustand übereinstimmen.
Ein Anwendungsbeispiel hierfür wäre die Drosselung eines Windkraftwerks auf die halbe Kraft. Über das Kommunikationsmodul würde die Steuerungslogik ein Ereignis von der *Central Grid Management Unit* (CGMU) zugesendet bekommen. Danach wechselt sie in den Zustand „halbe Kraft“ und aktualisiert den Produktionswert des Modells.
Bei Änderung von Werten bzw. des Zustandes veranlasst die Steuerungslogik auch den Versand von Ereignissen die die Änderungen der CGMU anzeigen.

Kommunikationsmodul

Die durch das Kommunikationsmodul angebotene Schnittstelle *ICommunicationModule* spezifiziert die Menge an Ereignissen, die ein Netzteilnehmer aussenden kann und beschreibt den Teilnehmer aus Sicht der ereignisgesteuerten Architektur. Zusätzlich ist das Modul für die Benachrichtigung der Steuerungslogik über eingegangene Ereignisse verantwortlich.
Hierzu führt es den folgenden Ablauf durch:

1. Das Kommunikationsmodul wird durch den Kommunikationsadapter über ankommende Nachrichten informiert.
2. Das Modul prüft, ob das enthaltende Ereignis für den Netzteilnehmer bestimmt ist und von ihm unterstützt wird.
3. Handelt es sich um ein valides Ereignis wird die Nachricht in eben dieses Ereignis umgewandelt; ansonsten wird sie verworfen.
4. Die Steuerungslogik wird über das Ereignis informiert.

Somit handelt es sich bei dem Kommunikationsmodul um die zentrale Komponente im Modell zur Filterung und Übersetzung von Ereignissen. Wohingegen die Verarbeitung von der Steuerungslogik übernommen wird.

6.2. Architektur des Monitorings

Beim Monitoring handelt es sich um ein sehr visuell geprägtes Teilsystem. Seine Aufgabe besteht in der Darstellung des zeitlichen Verlauf des Systemzustandes. Dieser kann aus den im System ausgetauschten Ereignissen extrahiert werden. Zur grafischen Darstellung des Systemzustandes werden unterschiedliche Sichten auf die Ereignisse benötigt. Daher wird die Systemarchitektur an eine gängige Architektur für Systemoberflächen angelehnt. Die Verwendung eines verbreiteten Entwurfsmusters, dem Model-View-Controller-Entwurfsmuster (MVC) (Fowler, 2002, S. 330), erschien für die Erfüllung der in Abschnitt 5.2.2 definierten Anforderungen am erfolgversprechendsten. Abbildung 6.2 zeigt das dazugehörige Komponentendiagram.

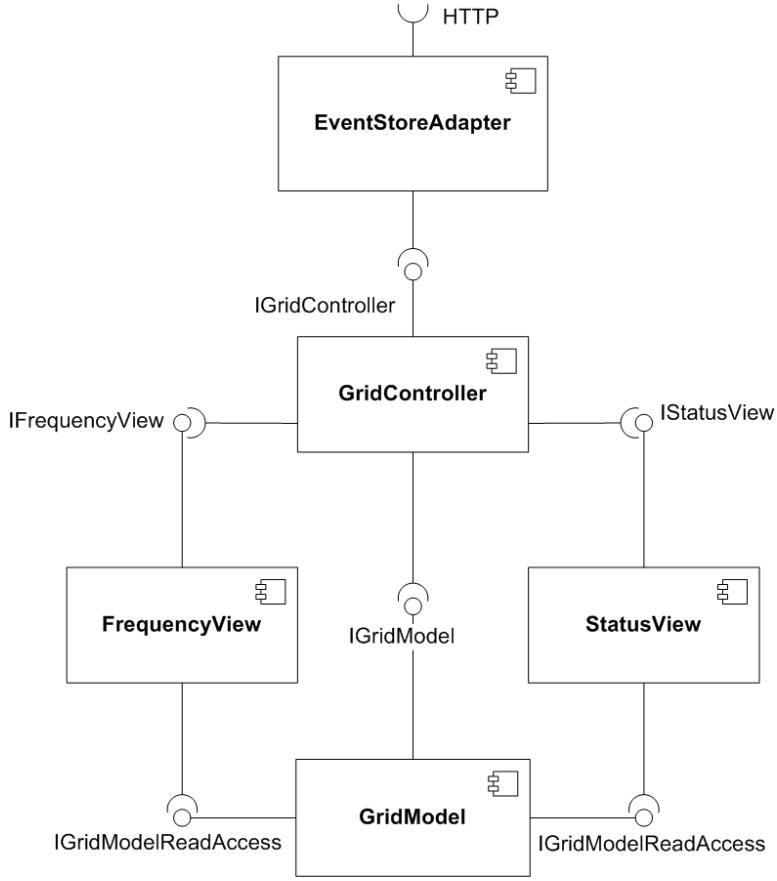

Abbildung 6.2.: Übersicht über die Architektur der Monitoringkomponente

6.2.1. Komponenten

Die Verbindung mit den anderen Teilsystemen wird beim Monitoring nicht über die Middleware realisiert. Da das Monitoring eine rein passive, beobachtende Rolle einnimmt, wird nur ein unidirek-

tionaler Kommunikationskanal benötigt. Deshalb wurde sich dafür entschieden die Verbindung über den *EventStore*, wie er bereits bei der technischen Architektur in Abschnitt 4.2 beschrieben wurde, vorzunehmen. Der *EventStoreAdapter* übernimmt die Kommunikation mit dem EventStore und hält über einen ereignisgesteuerten Mechanismus die Daten zur Bestimmung des Systemzustandes aktuell.

Hierauf baut nun die eigentliche MVC-Architektur auf. Das *GridModel* verwaltet die aktuellen und historischen Daten über den Netzzustand. Der *GridController* ist für die Steuerung und Koordination zwischen den einzelnen Komponenten zuständig. Er sorgt dafür, dass die Daten und die auf ihnen basierenden Sichten immer synchron sind. Der Ablauf sieht dabei wie folgt aus:

1. Ein neues Ereignis wird im *EventStore* gespeichert.
2. Der *EventStore* benachrichtigt den *EventStoreAdapter*.
3. Der *EventStoreAdapter* gibt das Ereignis an den *GridController* weiter, der dieses dann verarbeitet.
4. Der *GridController* aktualisiert das *GridModel* und benachrichtigt die von der Änderung betroffenen *View-Komponenten*.

An dieser Stelle sei darauf hingewiesen, dass die letztendliche Implementierung durch die durchgängige Verwendung von Ereignissen anstelle von Methoden-Aufrufen eine Architektur von lose gekoppelten Komponenten ermöglicht. Im Gegensatz zu dem während der Planung beschrittenen klassischem Ansatz wurde dadurch eine höhere Flexibilität, Robustheit und leichtere Veränderbarkeit erreicht.

6.3. Architektur der Central Grid Management Unit

Die hier vorgestellte Architektur der *Central Grid Management Unit* (CGMU) kann in dieser Form auch auf die *Home Grid Management Unit* (HGMU) übertragen werden. Da sie, wie im Abschnitt 4.1 beschrieben, als zusätzliche Indirektionsebene fungiert und die selben Aufgaben wie die CGMU auf der Hausnetzebene übernimmt, liegt die Wiederverwendung der CGMU Architektur hier nahe.

Beim Entwurf wurde sich an den in Abschnitt 5.2 und 5.4 beschriebenen Anforderungen orientiert. Zur Erfüllung der Anforderungen wurden die in der Abbildung 6.3 dargestellten Komponenten definiert. Für zukünftige Weiterentwicklungen, in Richtung des Einsatzes von Methoden des Data-Minings bzw. der Predictive Analytics, ist die Speicherung der Ereignisse zur Analyse direkt in der CGMU von Vorteil. Daher wurde in der in Abbildung 6.3 dargestellten Architektur bereits eine *EventStore*-Komponente vorgesehen. Diese muss konsequenterweise nicht identisch mit dem in der technischen Infrastruktur beschriebenen *EventStore* sein (vgl. Abschnitt 4.2).

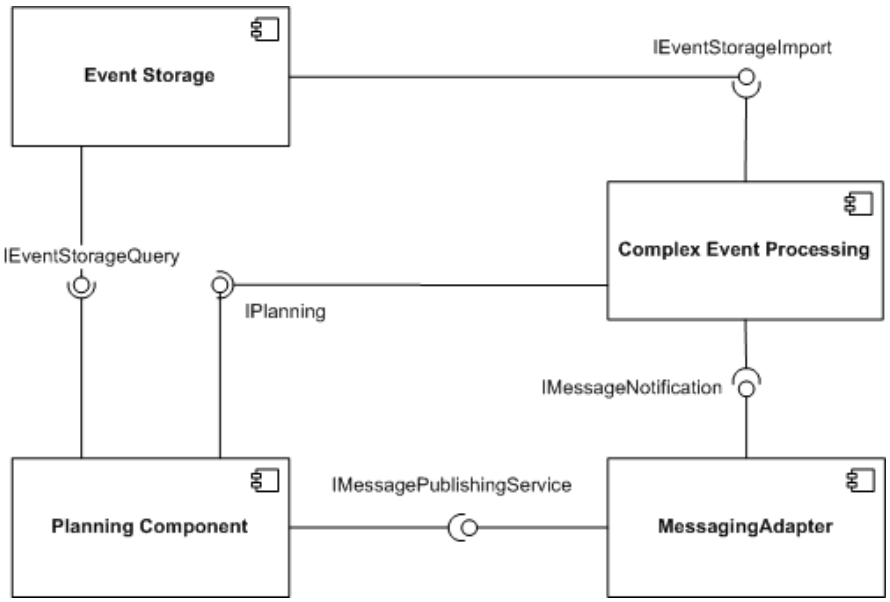

Abbildung 6.3.: Architektur der *Central Grid Management Unit*

Die Verbindung mit den anderen Teilsystemen, insbesondere der Simulationsumgebung, erfolgt auch hier über einen zentral bereitgestellten *MessagingAdapter*. Die *Complex Event Processing*-Komponente führt die eigentliche Ereignisverarbeitung durch. Sie wertet die ankommenden Ereignisse aus und stößt, in Abhängigkeit vom ermittelten Zustand des Stromnetzes, die Planung von Steuerungsmaßnahmen an.
Für die Planung ist die *Planning*-Komponente zuständig. Sie ermittelt einen Fahrplan an Aktionen zur Stabilisierung der Netzfrequenz und veranlasst deren Durchführung, indem sie über den *MessagingAdapter* die korrespondierenden Ereignisse an die Netzteilnehmer sendet.

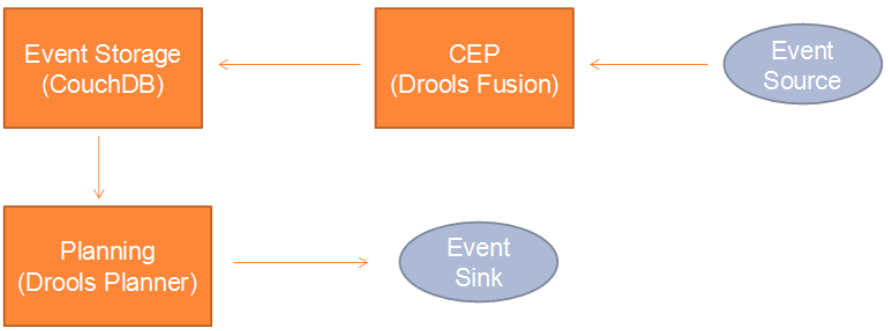

Abbildung 6.4.: Ereignisfluss durch die Architektur der *Central Grid Management Unit*

Somit gestaltet sich der Ereignisfluss innerhalb der CGMU wie in Abbildung 6.4 dargestellt. Die eingehenden Ereignisse werden zur Zustandsermittlung genutzt und im Anschluss persistiert. Auf ihnen aufbauend werden die Planungsschritte ermittelt und anschließend ausgeführt. Als Ergebnis

werden neue Ereignisse zur Steuerung der Netzteilnehmer an selbige versandt.

6.4. Zusammenfassung

In diesem Kapitel wurden die Architekturentwürfe der unterschiedlichen Teilsysteme vorgestellt und ihre Entstehung erläutert. Die einzelnen Teilsysteme realisieren im Verbund den Funktionsumfang des Gesamtsystems. Die Architektur des Gesamtsystems sieht somit folgendermaßen aus:

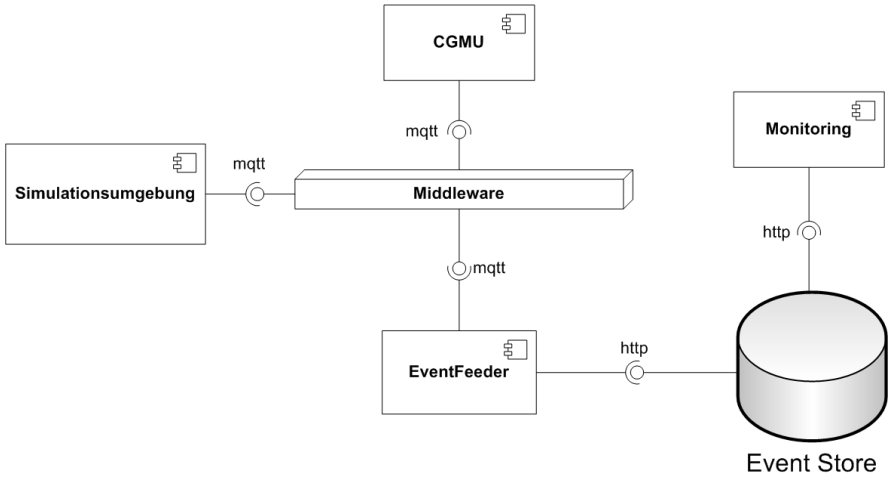

Abbildung 6.5.: Architektur des Gesamtsystems

Durch die konsequente Konzentration auf Ereignisse zur Kommunikation und Steuerung, sowie den Einsatz von Technologien, die dies unterstützen, konnte eine vollständig ereignisgesteuerte Architektur geschaffen werden. Sowohl die Kommunikation zwischen Teilsystemen als auch der Großteil der internen Architektur der Teilsysteme wird über Ereignisse geregelt.
Von Vorteil ist die hierdurch erreichte lose Kopplung der Komponenten. Die Komponenten sind robust gegenüber Ausfällen von anderen Komponenten und sind sehr gut isoliert testbar.
Der Nachteil einer solchen Architektur liegt in der aufwändigeren und schwierigeren Fehlersuche. Diese kann durch ein konsequentes testgetriebenes Vorgehen aber deutlich vereinfacht werden, indem die isolierte Testbarkeit der Komponenten ausgenutzt wird.

Eine über alle Ebenen erfolgende Ereignissteuerung führt zu der Entwicklung einer *Micro Service Architecture*. Die beschriebenen Vor- und Nachteile treffen auch auf den Einsatz von *Micro Services* zu. Für eine ausführliche Betrachtung dieses Paradigmas sei an dieser Stelle auf den Blog-Eintrag von James Hughes[1] verwiesen. Dieser stellt den Ansatz detailliert vor und betrachtet auch die Vor- und Nachteile, die der Einsatz einer solchen Architektur mit sich bringt.

[1] `http://yobriefca.se/blog/2013/04/29/micro-service-architecture/`

7. Entwurf

Das Ziel dieses Kapitels liegt in dem Entwurf von Modellen zur Beschreibung der zentralen Problemstellungen. Hierzu wird zunächst, aufbauend auf den in Abschnitt 5.2.3 formulierten Anforderungen, das zentrale Ereignismodell entworfen. Es beschreibt die Kommunikation zwischen den Teilsystemen und welche Informationen die Central Grid Management Unit (CGMU) zur Erfüllung ihrer Aufgaben benötigt.
In Abschnitt 7.2 wird dann das, der Netzsteuerung zugrundeliegende, Planungsproblem allgemein beschrieben. Aufbauend auf der Beschreibung wird abschließend das Modell des Planungsproblems entwickelt.

7.1. Ereignismodell

Ein Ereignismodell beschreibt, welche Ereignisse innerhalb eines Systems existieren und wie sie sich ggf. hierarchisch zusammensetzen. Zusätzlich kann beschrieben werden, wann ein Ereignis ausgelöst wird und mit welchen Regeln oder Algorithmen es verarbeitet wird (Luckham und Schulte, 2013, S. 2,3).
Dieser Abschnitt stellt das in dieser Arbeit entwickelte Ereignismodell vor. Neben den einzelnen Ereignissen und ihren Funktionen werden auch die konkreten Situationen, in welchen ein Ereignis ausgelöst wird, beschrieben. Da die Kommunikation aller Teilsysteme auf dem hier beschriebenen Ereignismodell beruht, erhält man auf diese Weise eine Übersicht über die grundlegenden Abläufe innerhalb des Gesamtsystems.

7.1.1. Basisereignisse

Auf Grundlage der in Abschnitt 5.2.3 beschriebenen Anforderungen an die *Central Grid Management Unit* (CGMU) wurden die in Abbildung 7.1 dargestellten Ereignisse ermittelt. Sie bilden die Ereignisbasis, um den Zustand des Stromnetzes erfassen zu können.

Aufbauend auf der in Steudte (2011) realisierten Ereignishierarchie, wurden in dieser Arbeit ein identisches Vorgehen angewandt. Als Erweiterungs- und Ausgangspunkt aller Ereignisse dient die abstrakte Klasse *BaseEvent*. Sie kapselt die minimale Funktionalität und Anforderungen, welche an alle Ereignistypen gestellt werden. Dieses sind im Einzelnen:

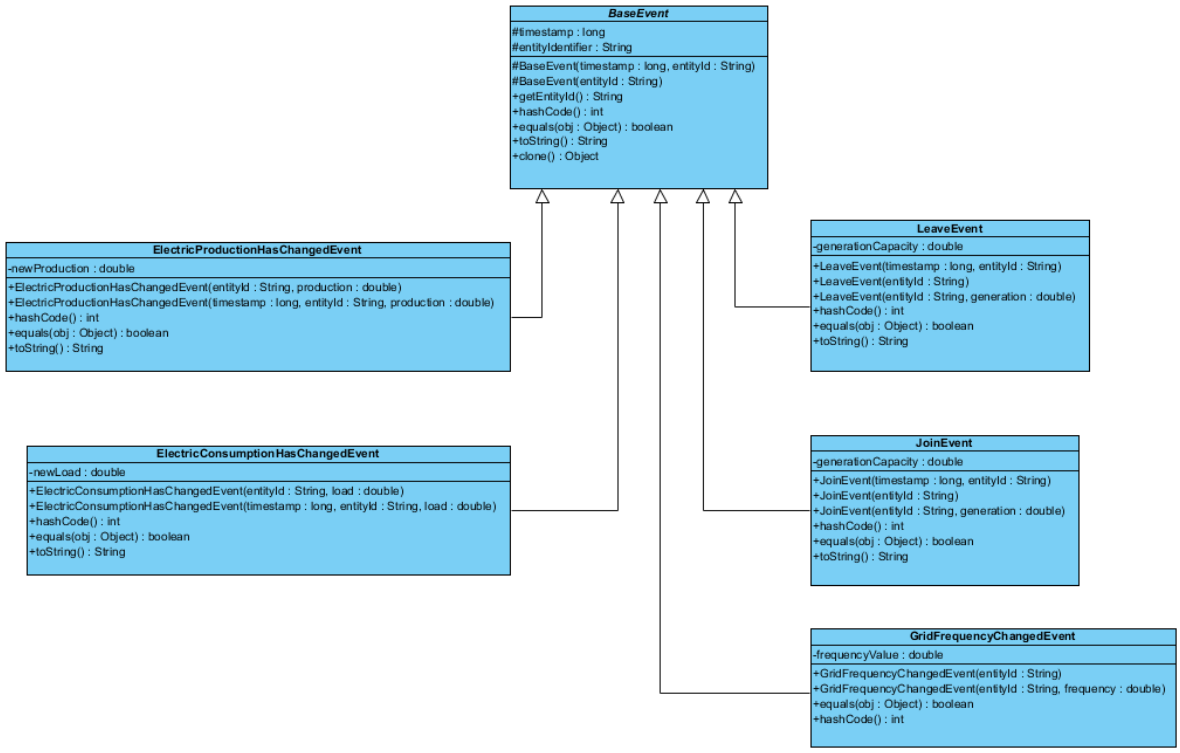

Abbildung 7.1.: Grundlegende Ereignisse zur Kommunikation zwischen Netzteilnehmern und CGMU.

- Bereitstellung eines Zeitstempels um Schlüsse über die zeitliche Abfolge von Ereignissen ziehen zu können.
- Den Bezeichner der Ereignisquelle zur Verfügung stellen.
- Die Basisfunktionalität für die zur Verarbeitung benötigten Methoden *hashCode()* und *equals()* bereitstellen.

So kann die Klasse *BaseEvent* als Blaupause für die Erzeugung neuer Ereignistypen genutzt bzw. betrachtet werden.

Join- und *LeaveEvents* signalisieren den Beitritt bzw. das Verlassen des Netzes eines Teilnehmers. Das *JoinEvent* wird als erstes Ereignis nach dem Anschluss des Teilnehmers an das Netz gesendet. Das *LeaveEvent* ist das letzte Ereignis, welches an die CGMU gesendet wird, bevor der Teilnehmer vom Netz getrennt wird.
Dabei wird der CGMU die maximal mögliche Erzeugungskapazität (in Watt) durch das Attribut *generationCapacity* mitgeteilt. Die Erzeugungskapazität wird benötigt, um die interne Trägheit des Netzes, bezogen auf die Frequenzveränderung, zu berechnen. Sie wird zur Berechnung der Netzfrequenz benötigt und bei Netzteilnehmern, die nur als Verbraucher arbeiten, mit $0{,}0\,\mathrm{W}$ angegeben.

ElectricProductionHasChangedEvent, *ElectricConsumptionHasChangedEvent* und *GridFrequencyChangedEvent* signalisieren direkte Veränderungen des Netzzustandes. Ersteres wird von einem Netzteilnehmer immer dann versendet, wenn sich seine Menge an produzierter Energie ändert und teilt damit den neuen Produktionswert mit. Das *ElectricConsumptionHasChangedEvent* erfüllt die gleiche Aufgabe auf Seiten des Energieverbrauchs.
Für die Anzeige der Netzfrequenzänderung wurde ein *GridFrequencyChangedEvent* vorgesehen. Dieses wird nur benötigt, wenn ein eigenständiges Teilsystem zur Ermittlung der Netzfrequenz genutzt werden sollte.

Die zentrale Idee hinter allen gerade beschriebenen Basisereignissen ist die Bekanntgabe der Veränderung eines Zustandes, in diesem Fall des Netzzustandes. Die Veränderung eines Zustandes entspricht der in der Fachliteratur geläufigen Definition eines Ereignisses, wie sie auch u.a. in Luckham (2001) sowie Etzion und Niblett (2010) verwendet wird.
Ereignisse bieten Vorteile gegenüber der periodischen Übermittlung etwa von Messdaten, da so die Anzahl an Ereignissen deutlich reduziert werden kann. Auch verringert sich der Rechenaufwand, wenn nur der aktuelle Zustand betrachtet werden muss. Bei der Betrachtung von Zeitverläufen sind allerdings periodische Daten von Vorteil, da bei gespeicherten Ereignissen hierzu zusätzliche Abfragen und eine aufwändigere Differenzrechnung benötigt werden. Bei der Implementierung des Monitoring-Teilsystems wurde daher nur eine Überwachung des Systems in Echtzeit implementiert und auf eine historische Betrachtung zunächst verzichtet.

7.1.2. Angebotsereignisse

Bei der Verarbeitung von Angebotsereignissen spielt die angesprochene Differenzberechnung eine wichtige Rolle. Zunächst soll aber die Funktionalität der Ereignisse im Mittelpunkt stehen.
Angebotsereignisse signalisieren der CGMU die Möglichkeit zur Steuerung des Netzteilnehmers. Sie repräsentieren Veränderungen des Zustandes eines Netzteilnehmers, die durch die CGMU ausgelöst werden können. Dabei geben die Ereignisse bekannt, um welchen Wert die produzierte oder verbrauchte Energiemenge geändert werden kann. Das *ProductionOfferEvent* signalisiert die Möglichkeit zur Beeinflussung der Energieproduktion, während das *ConsumptionOfferEvent* die Beeinflussungsmöglichkeit für die Angebotsseite darstellt.

BaseOfferEvent

Beide Ereignisse stellen dabei konkrete Ausprägungen der abstrakten Basisklasse *BaseOfferEvent* dar. Diese kapselt, wie auch schon bei den Basisereignissen, die für diese Klasse von Ereignissen wichtigen Attribute und dient somit auch hier als Erweiterungspunkt.
Das *changeValue*-Attribut teilt mit, um wieviel Watt die Energieproduktion bzw. der -verbrauch des Netzteilnehmers beeinflusst werden kann. Ein positives *changeValue*-Attribut stellt dabei eine

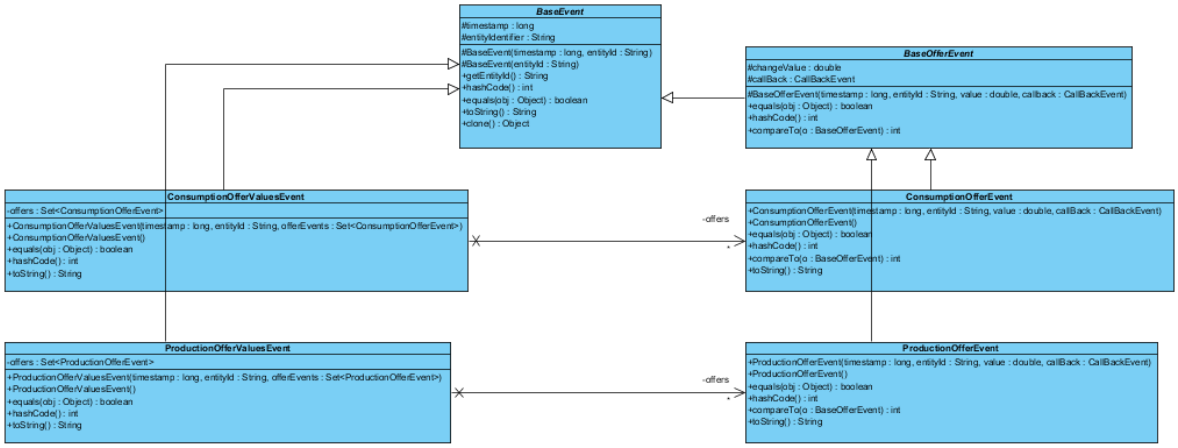

Abbildung 7.2.: Übersicht über die an Angebotsereignissen beteiligten Klassen

Erhöhung des aktuellen Wertes dar, wohingegen ein negativer Wert eine Drosselung bekanntgibt. Unter dem Attribut *callBack* wird das mit dem Angebotsereignis assoziierte *CallBackEvent* hinterlegt. Die Betrachtung von *CallBackEvents* erfolgt in Abschnitt 7.1.3. Es sei vorausgeschickt, dass sie von der CGMU dazu verwendet werden, um den Netzteilnehmer zu benachrichtigen den durch das Angebotsereignis repräsentierten Zustandswechsel vorzunehmen.

Ereignisversand

Angebotsereignisse werden in den folgenden Situationen versandt:

- Beitritt eines Netzteilnehmers zum Stromnetz.
- Bei Änderungen des inneren Zustandes eines Teilnehmers z.B.:
  - Änderung der Umweltbedingungen.
  - Erhalt eines *CallBackEvents*.

OfferValuesEvents

Diese Ereignisse stellen den Mechanismus zur Aktualisierung der in der CGMU vorgehaltenen Angebotsereignisse dar. Während der Realisierung konnten hierfür zwei mögliche Aktualisierungsmechanismen identifiziert werden:

1. Jedes *OfferEvent*, welches ein bereits versendetes ersetzt, muss das zu ersetzende Ereignis als Referenz enthalten.
2. Ein Sammelereignis, welches alle gerade gültigen Angebotsereignisse umfasst, so dass mittels einer Differenzrechnung die zu entfallenden und weiterhin gültigen Angebotsereignisse ermittelt werden können.

Bei der Implementierung wird die letztere Variante genutzt und mittels der *OfferValuesEvents* umgesetzt. Die Vorteile dieser Varianten liegen darin, dass:

1. Der Umstand berücksichtigt wird, dass bei einer Änderung des Netzteilnehmerzustandes in der Regel mehrere Angebotsereignissen gesendet werden müssen, da oftmals mehr als ein Folgezustand existiert.

2. Die Menge an ausgetauschten Ereignissen somit deutlich reduziert werden kann.

Die Differenzberechnung erfolgt dabei auf Basis der Mengentheorie und der Bildung von Differenzmengen. Sie soll anhand des folgenden Beispiels verdeutlicht werden.
Den Ausgangspunkt bildet eine Menge an Angebotsereignisse, welche durch ein *OfferValuesEvent* der CGMU mitgeteilt wurde:

$$OldOffers = \{-50, +50, +300, -1337, +299\} \tag{7.1}$$

Danach erreicht ein neues *OfferValuesEvent*, vom gleichen Typ und gleichen Netzteilnehmer, die CGMU:

$$NewOffers = \{-50, +50, +200, -200, +331\} \tag{7.2}$$

Um die Gesamtmenge an zur Verfügung stehenden Angebotsereignissen aktualisieren zu können, werden die neu hinzukommenden und die entfallenden Ereignisse benötigt. Die erste Menge wird im Folgenden mit $ToBeAdded$ und die zweite mit $ToBeRemoved$ bezeichnet.
Beide werden mittels der Differenzbildung zwischen den beiden Mengen $OldOffers$ und $NewOffers$ berechnet. Hinzugefügt werden müssen die Ereignisse welche in $NewOffers$ sind, aber nicht in $OldOffers$, also:

$$ToBeAdded = NewOffers \setminus OldOffers \tag{7.3}$$

Während dessen müssen die Ereignisse die nur in $OldOffers$ enthalten sind aus der Gesamtmenge an Angebotsereignissen entfernt werden. Somit ist die Menge $ToBeRemoved$ definiert als:

$$ToBeRemoved = OldOffers \setminus NewOffers \tag{7.4}$$

So ergeben sich aus dem Beispiel folgende Differenzmengen:

$$ToBeAdded = \{+200, -200, +331\} \tag{7.5}$$

$$ToBeRemoved = \{+300, -1337, +299\} \tag{7.6}$$

Mit diesen Mengen würde die CGMU dann die Gesamtmenge an Angebotsereignissen aktualisieren. Diese Methodik stellt die eigentliche Erweiterung des bereits in Steudte (2011) beschriebenen Ereignismodells dar. Bedeutung kommt nicht nur den einzelnen Ereignissen und ihrer zeitlichen Abfolge zu, sondern liegt sie auch in der Differenz zweier aufeinanderfolgender Ereignisse. Dieses wird als Mechanismus zur Aktualisierung der Ereignismengen explizit ausgenutzt.

7.1.3. CallBack-Ereignisse

Die zweite Erweiterung stellen die bereits angesprochenen *CallBackEvents* dar. Jedes *OfferEvent* besitzt ein passendes *CallBackEvent*, welches von CGMU zum Abrufen der Zustandsänderung genutzt werden kann.

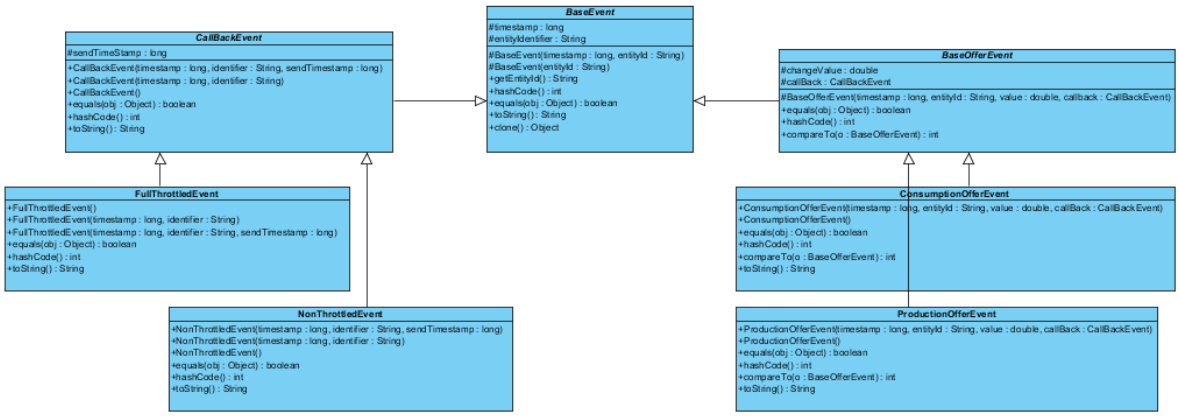

Abbildung 7.3.: Übersicht CallBack- und OfferEvent-Hierarchie

CallBackEvents leiten von der Basisklasse *CallBackEvent* ab. Sie erweitert die Klasse der Basisereignisse um ein Attribut *sendTimeStamp*, welches von der CGMU zum Zeitpunkt des Sendens des Ereignisses mit einem Zeitstempel versehen wird. Dieses war als Mechanismus zur groben Abschätzung der Bearbeitungszeit konzipiert, wird aber bei den in dieser Arbeit betrachteten Szenarien nicht benötigt. Seine Verwendung bietet sich allerdings zur Durchführung von Messungen sowie zum Debugging an.

CallBackEvents unterscheiden sich von allen bisher vorgestellten Ereignissen, da sie nach ihrer Erstellung noch verändert werden können. Alle anderen hier betrachteten Ereignisse sind immutable. Ihre Attribute sind final und es existieren keine öffentlichen Setter-Methoden. Sie können also nur per Konstruktoraufruf erzeugt und danach nicht mehr geändert werden.
Möchte man die Unveränderlichkeit auch bei *CallBackEvents* erhalten, müsste man den *sendTimeStamp* in ein zusätzliches zu sendendes Ereignis ausgliedern.

FullThrottled- und NonThrottledEvents

Die vorgenannten konkreten Ausprägungen von *CallBackEvents* gehen auf die Idee zurück, den inneren Zustand der Energieproduzenten mit Hilfe eines endlichen Automaten zu modellieren. Das *FullThrottledEvent* sorgt dafür, dass die Produktionsmenge des Netzteilnehmers auf $0,0\mathrm{W}$ gedrosselt wird. Das *NonThrottledEvent* sorgt dafür, dass der Netzteilnehmer in den Modus ohne Drosselung wechselt und wieder die ursprüngliche Energiemenge produziert. Sie stehen an dieser Stelle beispielhaft für alle weiteren im System denkbaren *CallBackEvents*, wie z.B. ein *HalfThrottledEvent* welches

die Produktion auf die Hälfte reduziert.

7.1.4. Zusammenfassung

Mit dem in diesem Kapitel beschriebenen Ereignismodell konnten die an die CGMU gestellten Anforderungen, bezogen auf die Überwachung und Steuerung des Netzzustandes, erfüllt werden. Auf Grund der dynamischer Steuerung des Stromnetzes fällt das Ereignismodell komplexer aus, als in der vorangegangenen Arbeit (vgl. Steudte (2011)). Dennoch ist es dank der definierten Basisklassen übersichtlich und flexibel erweiterbar.
Bei dem Entwurf und der Umsetzung des Ereignismodells wurde die Erfahrung gemacht, dass das Modell nicht nur von der eingesetzten Programmiersprache, sondern auch von den zugrundeliegenden Prinzipien des Anwendungsbereichs abhängt. So war es in diesem Fall nötig einen komplexeren Mechanismus zu entwickeln, um Ereignisse widerrufen zu können. In anderen Anwendungsbereichen könnte dies keine Rolle spielen. Daher sind die in den Ereignismodellen verwendeten Grundprinzipien nicht ohne Weiteres auf andere Anwendungsbereiche übertragbar.

7.2. Planungsmodell

Die Lösung des Planungsproblems, wie Angebot und Nachfrage nach Energie einander am sinnvollsten angeglichen werden können, stellt die grundlegende Aufgabe bei der Steuerung eines Stromnetzes dar. Werden heutzutage Lastpläne sowie manuelle Eingriffe zur Lösung des Problems genutzt, so ist zukünftig eine voll automatisierte Lösung des Planungsproblems denkbar.
Für eine automatisierte Lösung muss das zu lösende Planungsproblem zunächst genau erfasst und beschrieben werden (vgl. Abschnitt 7.2.1). Die Problembeschreibung muss dann im Anschluss in eine, dem Computer verständliche, Repräsentation des Problems überführt werden.
Da in dieser Arbeit das Framework *Jboss Opta Planner* zur Lösung des Planungsproblems genutzt wird, findet in Abschnitt 7.2.2 die Überführung der Problembeschreibung in ein Modell mit Hilfe der durch *Jboss Opta Planner* bereitgestellten Hilfsmittel und Modelle statt. Für eine genaue Betrachtung der Für und Wider der Nutzung eines Frameworks zur Lösung von Planungsproblemen, sei an dieser Stelle auf den Abschnitt 8.2.3 verwiesen.

7.2.1. Problembeschreibung

Das eigentliche Ziel, welches durch die CGMU zu verfolgen ist, leitet sich aus der in Abschnitt 1.4 vorgestellten Hypothese *H3* ab. Sie soll die Teilnehmer des Netzes so automatisiert Steuern, dass ein stabiler Netzbetrieb sichergestellt ist. Vereinfacht bedeutet ein stabiler Betrieb, dass sich zu jedem Zeitpunkt die Mengen an angebotener und nachgefragter Energie ausgleichen müssen. Um das Ziel zu erreichen, ist das eigentliche Planungsproblem zu lösen. Dieses kann folgendermaßen beschrieben werden:

> In Abhängigkeit von den aktuellen Produktions- und Verbrauchswerten sollen aus der Menge der OfferEvents die jeweiligen Ereignisse bestimmt werden, so dass Energieangebot und -nachfrage wieder ausgeglichen sind.

Dieses Planungsproblem automatisiert zu lösen, ist die Aufgabe der CGMU und insbesondere ihrer Planungskomponente.

7.2.2. Modellierung des Planungsproblems

Um das beschriebene Planungsproblem automatisiert lösen zu können, muss es in eine computerverständliche Repräsentation überführt werden. Hierfür bietet *Opta Planner* zwei Arten von Klassen an:

1. Problem Facts
2. Planning Entities

Problem Facts sind die Einheiten, die zur Lösung eines Planungsproblems genutzt werden können. In dem in dieser Arbeit betrachteten Anwendungsbereich handelt es sich dabei um alle *OfferEvents*, also sowohl *Consumption-* als auch *ProductionOfferEvents*. Sie stellen die Konstanten bei der Lösung des Problems dar und verändern sich während der Lösung des Planungsproblems nicht. Für unseren Anwendungsbereich bedeutet das, dass sobald neue *OfferEvents* an der CGMU ankommen, es sich um ein neues Planungsproblem handelt und somit eine erneute Planung nötig wird. Dieser Vorgang wird als *Real-Time Planning* bezeichnet.
Planning Entities, im Gegensatz zu den *Problem Facts*, sind die Teile eines Problems die sich während dessen Lösung verändern. Diese können im hier vorliegenden Anwendungsfall nicht direkt aus der Problembeschreibung abgeleitet werden.

Bestimmung der Planning Entities

Um eine geeignete Repräsentation zu finden, konzentriert sich die Modellierung auf das Ergebnis eines Planungsvorgangs. Als Ergebnis eines erfolgreichen Planungsvorgangs soll eine Liste von *OfferEvents* vorliegen. Diese sollen das Netz, unter Zuhilfenahme der assoziierten CallBack-Ereignisse, in einen stabilen Zustand überführen. Das Ergebnis stellt also einen Fahrplan zur Netzsteuerung dar. Aus dieser Überlegung ging die Idee hervor, dass die Einheiten, die während des Planungsprozesses verändert werden, die Belegungen der Plätze innerhalb der Liste bzw. des Fahrplans sind.
Das in Abbildung 7.4 dargestellte Klassendiagramm veranschaulicht die letztendliche Modellierung des Planungsproblems. Die Klasse *ScheduleItem* repräsentiert dabei einen zu füllenden Platz innerhalb des Fahrplans. Sie stellt damit ein *Planning Entity* dar.
Das Attribut *item* der Klasse *ScheduleItem* agiert als Planungsvariable und wird durch das Framework in den Planungsschritten mit Planungswerten belegt. Planungswerte sind die *Problem Facts* und hier

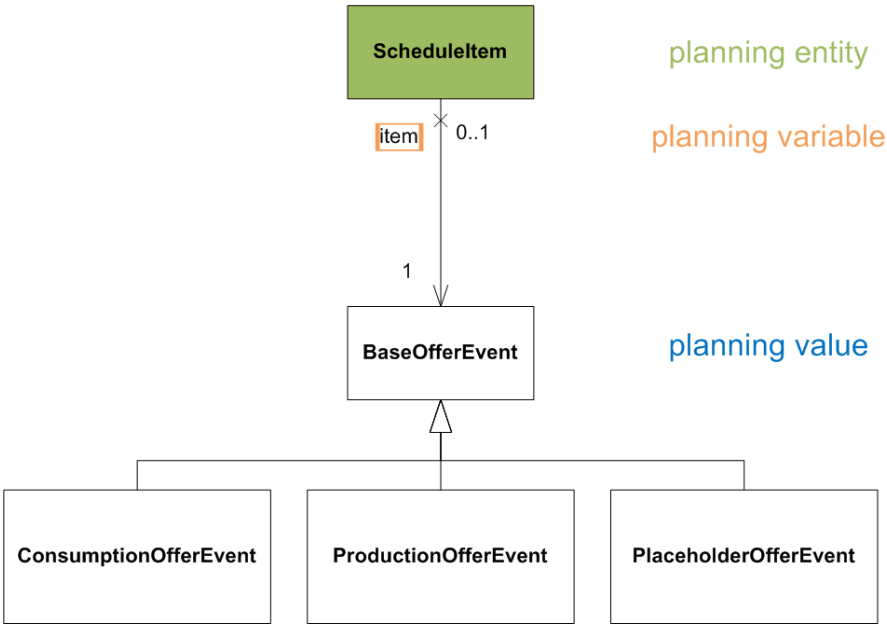

Abbildung 7.4.: Klassendiagramm des Planungsproblems

im speziellen die *OfferEvents* der Netzteilnehmer. Sie werden durch die Basisklasse *BaseOfferEvent* und ihre einzelnen Ausprägungen repräsentiert. Die Tabelle 7.1 vermittelt eine zusammenfassende Übersicht über die verwendeten Klassen und ihre Funktion bei der Modellierung des Planungsproblems.

Klasse	Funktion im Framework	Planungsfunktion
ScheduleItem	Planning Entity	Planning Entity
BaseOfferEvent	Planning Value	Problem Fact
ConsumptionOfferEvent	Planning Value	Problem Fact
ProductionOfferEvent	Planning Value	Problem Fact
PlaceholderOfferEvent	Planning Value	Problem Fact

Tabelle 7.1.: Planungsklassenübersicht und Funktionszuordnung

7.2.3. Zusammenfassung

Ausgehend von der Problembeschreibung, Angebot und Nachfrage stets auszugleichen, konnte in diesem Abschnitt ein Modell des Planungsproblems entworfen werden. Dieses Modell beschreibt das Problem, mit Hilfe der durch *Jboss Opta Planner* angebotenen Hilfsmittel, als die Bestimmung eines Fahrplans an OfferEvents zum Angleichen von Angebot und Nachfrage. Auf dieser Beschreibung aufbauend, kann im nächsten Kapitel die Realisierung des Planungsprozesses mit Hilfe von *Jboss Opta Planner* erfolgen.

8. Realisierung

In diesem Kapitel soll die Umsetzung der einzelnen Teilsysteme beschrieben werden. Auf Grund des Umfangs beschränken sich die Erläuterungen auf die zentralen oder besonders interessanten Teilsysteme und Aspekte. Eine vollständige Betrachtung alle Teilsysteme würde den vorgegebenen Rahmen einer Masterarbeit überschreiten.
Zum Einstieg in dieses Kapitel wird in Abschnitt 8.1 die Umsetzung der Kommunikation, über alle beteiligten Ebenen hinweg, vorgestellt. Dabei wird besonders auf die Umwandlung der Ereignisse in ein transportfähiges Format und die Schwierigkeiten bei der Implementierung des Kommunikationsadapters auf Seiten der Simulationsumgebung eingegangen.
Anschließend wird die Umsetzung des Ereignismodells aus Abschnitt 7.1, sowohl in Java als auch in C#, beschrieben.
Die Betrachtung der Implementierung der *Central Grid Mangement Unit* (CGMU) findet in Abschnitt 8.2 statt. Dabei wird beschrieben, wie die Netzsteuerung mit Hilfe von *Jboss Opta Planner* umgesetzt wurde und welche Prinzipien zum Einsatz kamen.
Danach werden in den Abschnitten 8.3 und 8.4 die zentralen Eckpunkte bei der Umsetzung der Simulationsumgebung und des Monitoringsystems vorgestellt. Für eine ausführlichere Vorstellung der beiden Systeme und ihrer Entwicklung, soll an dieser Stelle auf Steudte (2013) verwiesen werden.

8.1. Kommunikationsinfrastruktur

Wie bereits im Abschnitt 4.2 berichtet wurde, erfolgt die Kommunikation zwischen den Teilsystemen über MQTT. Zur letztendlichen Realisierung der Infrastruktur bedurfte es noch der Auswahl eines *Message Brokers*, der MQTT als Transportprotokoll unterstützt. Zusätzlich waren *Kommunikationsadapter* auf Seiten der Simulationsumgebung, der CGMU und des Event Feeders zu implementieren. Die beiden letzteren wurden dabei in Java umgesetzt, während die Simulationsumgebung aus den angesprochen Gründen der Portabilität mit .NET und C# realisiert wurde. Daher musste insgesamt nur ein Adapter für die CGMU und den Event Feeder umgesetzt werden. Hierbei konnte auf die in Steudte (2011) realisierten Komponente aufgesetzt werden.

8.1.1. Serialisierung und Deserialisierung von Ereignissen

Um die Ereignisobjekte übertragen zu können, müssen sie in eine als Nachricht übertragbare Repräsentation überführt und nach dem Erhalt zurückgewandelt werden. Im Regelfall kommen als

Repräsentation Zeichenketten oder Byte-Arrays in Frage. Beide Repräsentationen lassen sich bei Bedarf ineinander konvertieren.
Auf Grund der positiven Erfahrungen, die mit dem Einsatz von JSON[1] als Format zum Datenaustausch in Steudte (2011) gemacht wurden, kam es auch in dieser Arbeit zur Anwendung. Die Vorteile von JSON liegen darin, dass es einfach maschinell zu erzeugen und zu verarbeiten ist. Trotzdem kann es von Menschen gelesen und verstanden werden. Diese Eigenschaften erleichtern das Debugging der Kommunikation beträchtlich.
Zur Optimierung der Übertragung im Hinblick auf Effizienz und Machine-To-Machine-Kommunikation würde sich, nach erfolgreicher Realisierung der Kommunikationsabläufe, ein Umstieg auf Protocol Buffers[2] anbieten. Dabei büßt man zwar die Lesbarkeit durch Menschen ein, erhält dafür aber eine schnellere Umwandlung und kleinere Nachrichten, welche auch durch eingebettete Systeme effizient verarbeitet werden können.

Realisierung

Die Umwandlung der Ereignisse erfolgt in jedem Teilsystem durch die in Abbildung 8.1 dargestellten Komponenten. Dabei wird jeweils eine Komponente für die Umwandlung der Ereignisse in das Nachrichtenformat (*EventMessageTransformer*) genutzt. Eine weitere übersetzt ankommende Nachrichten in Ereignisse (*MessageEventTransformer*).

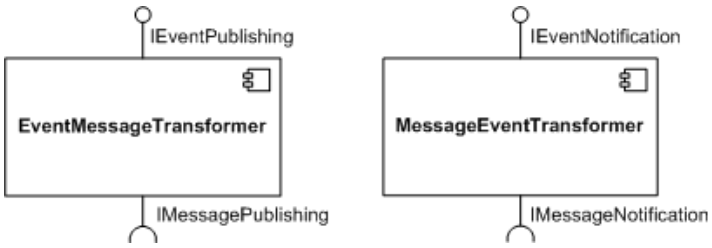

Abbildung 8.1.: Übersicht Transformer-Komponenten

Durch die angebotenen Schnittstellen und die Verwendung der durch den Kommunikationsadapter bereitgestellten Schnittstellen, lässt sich die Implementierung der Transformer-Komponenten jederzeit austauschen. Hierdurch ist eine einfache Änderung des Nachrichtenformats möglich.
Zur Umsetzung der Komponenten wurde auf die Bibliotheken *JSON.NET*[3] und *Google Gson*[4] zurückgegriffen. Beide Bibliotheken arbeiten durch ihre Konfigurierbarkeit über Annotationen sehr gut zusammen und schaffen es somit die .NET- und Java-Plattform miteinander zu verbinden.

[1] `http://www.json.org`
[2] `https://code.google.com/p/protobuf/`
[3] `http://james.newtonking.com/projects/json-net.aspx`
[4] `https://code.google.com/p/google-gson/`

8.1.2. Probleme mit der Kommunikation in .NET

Während der Unit- und Integration-Tests kam es zu Verbindungsabbrüchen zwischen dem eingesetzten *Really Small Message Broker*[5] und der Simulationsumgebung. Als Grund hierfür wurde zunächst der Message Broker vermutet. Nach Tests mit der Standardimplementierung eines Message Brokers für MQTT, *Mosquitto*[6], stand jedoch fest, dass der Fehler bei der verwendeten MQTT-Implementierung für .NET liegen musste.
Hierbei wurde zunächst die Bibliothek *MqttDotNet*[7] verwendet, da die zum Entwicklungszeitpunkt bereitstehende Alternative *nMQTT*[8] schon längere Zeit nicht mehr gewartet wurde. Letztendlich konnte eine stabile Kommunikation erreicht werden, nachdem der Message Broker auf *Apache ActiveMQ*[9] umgestellt wurde und der Kommunikationsadapter der Simulationsumgebung die *NMS-API*[10] zur Kommunikation mit dem Broker nutzte. ActiveMQ unterstützt auch MQTT, so dass außer der Simulationsumgebung keine anderen Teilsysteme angepasst werden mussten.
Ausgehend von der so etablierten technischen Kommunikation zwischen den Teilsystemen konnte ein Modell zur Kommunikation über fachliche Ereignisse entwickelt werden. Dieses orientiert sich an den fachlichen Anforderungen des Systems und wird im kommenden Abschnitt vorgestellt.

8.2. Central Grid Management Unit

Der CGMU kommt als zentrale Steuerungseinheit auch eine besondere Bedeutung innerhalb des Systems zu. Daher liegt der Schwerpunkt dieses Kapitels auch auf der Beschreibung ihrer Implementierung.
Dazu soll zu Beginn auf die Verarbeitung der Ereignisse durch die CGMU eingegangen werden, da sie die Grundlage für alle Abläufe innerhalb der CGMU darstellt. Anschließend wird in Abschnitt 8.2.3 erläutert, wie das in Abschnitt 7.1 entwickelte Ereignismodell in Java umgesetzt wurde. Danach wird das Modell dazu genutzt, um die Planungsabläufe mittels *Jboss Opta Planner* zu automatisieren und so eine Steuerung des Stromnetzes zu realisieren. Die hierbei verwendeten Ansätze und Prinzipien werden sowohl fachlich als auch in ihrer technischen Umsetzung erläutert.

8.2.1. Implementierung des Ereignismodells

Das im vorangegangenem Kapitel in Abschnitt 7.1 entwickelte Ereignismodell stellt eine Schnittstelle zwischen den Teilsystem dar. Daher ist es in allen Teilsystemen zu implementieren und wo möglich als Paket, z.B. als Jar-Archiv, auszuliefern. Auf diese Weise kann es automatisiert wiederverwendet

[5] `https://www.ibm.com/developerworks/community/groups/service/html/communityview?communityUuid=d5bedadd-e46f-4c97-af89-22d65ffee070`
[6] `http://mosquitto.org/`
[7] `https://github.com/stevenlovegrove/MqttDotNet`
[8] `https://github.com/markallanson/nmqtt`
[9] `http://activemq.apache.org/`
[10] `NMS`

werden.

Da die Simulationsumgebung mit .NET und C#, und damit auf einer anderen Plattform als die restlichen Teilsysteme realisiert wird, ist jeweils eine unabhängige Implementierungen pro Plattform zu erstellen. Auf beiden Plattformen wurde das Ereignismodell, wie in Abschnitt 7.1 dargestellt, als Klassenhierarchie umgesetzt. Unter Java wurden hierzu POJOs genutzt, unter C#, dem gleichen Ansatz folgend, waren es ebenso Standardklassen.
Die eigentliche Verbindung der beiden Modelle erfolgt, wie im Abschnitt 8.1.1 bereits beschrieben, über die eingesetzten Frameworks zur Serialisierung und Deserialisierung.
Durch sie werden die Namen der Attribute in der JSON-Repräsentation vereinheitlicht. Listing 8.1 und Listing 8.2 zeigen die Anpassungen für die Klasse BaseEvent in C# und Java.

```java
public abstract class BaseEvent implements Cloneable{

  /**
   * Timestamp in milliseconds when the event has been created.
   */
  @SerializedName("time_stamp")
  protected final long timestamp;

  /**
   * The id of the grid entity that has created the event.
   */
  @SerializedName("entity_id")
  protected final String entityIdentifier;

      ...
```

Listing 8.1: BaseEvent in Java

```csharp
[JsonObject(MemberSerialization.Fields)]
   public abstract class BaseEvent
   {
       /// <summary>
       /// Time object holding the unix time
       /// </summary>
       [JsonIgnore]
       static readonly DateTime StartOfEpoch = new DateTime(1970, 1, 1, 0, 0, 0,
           System.DateTimeKind.Utc);

       /// <summary>
       /// The Time when the event has happened
       /// </summary>
```

```
[JsonProperty(PropertyName = "time_stamp")]
protected readonly long timestamp;

/// <summary>
/// appliance which has send the event
/// </summary>
[JsonProperty(PropertyName = "entity_id")]
protected readonly string entityIdentifier;

...
```

Listing 8.2: BaseEvent in C#

Zusätzlich wurden Funktionen zur Umwandlung von C# DateTime-Objekten in Java-Millisekunden implementiert.
Durch diese Anpassungen konnten die Lücke zwischen den Plattformen JVM und .NET überwunden und die Teilsysteme miteinander verbunden werden.

8.2.2. Ereignisverarbeitung

Die Ereignisverarbeitung findet in der CGMU nur sehr rudimentär statt. Sie wird aktuell nur dazu genutzt, um auf die ankommenden Ereignisse zu reagieren. In Abhängigkeit vom Typ des Ereignisses wird die passende Handler-Methode in der CGMU aufgerufen. Diese übernimmt dann die eigentliche Verarbeitung des Ereignisses.

Auf diese Weise wird durch den Einsatz von Jboss Drools Java um die Fähigkeit erweitert Pattern Matching auf Objekten durchzuführen. Die Alternative wäre der Einsatz einer Programmiersprache, die diese Fähigkeit von Haus aus unterstützt. Ein Beispiel auf der JVM wäre hierfür Scala, welche Pattern Matching über ihre *Case Classes* realisiert (Odersky u. a., 2008, S. 293).

Jboss Drools unterstützt darüber hinaus bei der zukünftigen Umsetzung von erweiterten Steuerungsmöglichkeiten. Mit dem Modul *Drools Fusion* wurden erweiterte Fähigkeiten zum Complex Event Processing eingeführt. Hierzu gehört unter anderem *Temporal Reasoning*, als das Schlussfolgern über zeitliche Zusammenhänge zwischen Ereignisse.
Diese Fähigkeit, in Verbindung mit der Erweiterung der Steuerung um Methoden der *Predictive Analytics* bzw. des Data Mining, machen den Einsatz von Drools interessant, um eine vorausschauende Steuerung von Stromnetzen zu erforschen. An dieser Stelle ergibt sich ein möglicher Ansatzpunkt zur Fortführung dieser Arbeit.

8.2.3. Planung

Aufgabe der CGMU ist es, Maßnahmen zur Netzsteuerung anhand des Netzzustandes zu planen und anschließend durchzuführen.
Diese Aufgabe zu automatisieren ist nicht trivial:

1. Das System muss bei der Planungsdurchführung und der anschließenden Ausführung sehr schnell sein, da sich die Bedingungen im Netz, und damit das Planungsproblem, sehr schnell ändern können.

2. Planungsprobleme sind NP-vollständig, so dass eine optimale Lösung unter Umständen nicht in endlicher Zeit gefunden werden kann.

3. Es existieren verschiedene Algorithmen zur Lösung unterschiedlicher Planungsprobleme. Es ist nicht von Vornherein ersichtlich, welcher Algorithmus am geeignetsten ist, um das konkrete Planungsproblem zu lösen.

Das Frameworks *Jboss Opta Planner*[11] unterstützt bei der Lösung der gerade beschriebenen Probleme und nimmt dem Entwickler einen erheblichen Teil des Aufwandes bei der Realisierung ab.
Es bietet dazu folgende Hilfsmittel an:

+ Eine einheitliche Struktur zur Modellierung der Probleme wird vorgegeben, so dass verschiedene Algorithmen auf eine Problemstellung angewendet werden können.

+ Es ermöglicht die externe Konfiguration der Parameter der Algorithmen mittels XML.

+ Es liefert getestete Implementierungen gängiger Planungsalgorithmen aus.

+ Enthalten ist ein integriertes Benchmark-Werkzeug zum Vergleich unterschiedlicher Konfigurationen und Planungsalgorithmen.

Während der Nutzung von *Opta Planner* konnten die folgenden Nachteile identifiziert werden:

- Aufwändige Problemmodellierung auf Grund der vorgegebenen Struktur.

- Auf Grund des Funktionsumfangs hoher Einarbeitungsaufwand.

Zusammenfassend lässt sich jedoch ein sehr positives Fazit über den Einsatz von *Opta Planner* ziehen. Durch den Einsatz des Frameworks konnte der Aufwand für die Implementierung und den Test der verschiedenen Planungsalgorithmen deutlich reduziert werden. Außerdem kann man in relativ kurzer Zeit qualifizierte Aussagen über die Eignung der unterschiedlichen Algorithmen erhalten. Dass die Modellierung an einigen Stellen hierbei nicht ganz intuitiv ist, lässt sich in Anbetracht der Zeitersparnis und der anderen Vorzüge verschmerzen.

[11] `http://http://www.optaplanner.org/`

Implementierung des Planungsmodells

Das in Abschnitt 7.2 beschriebene Planungsmodell konnte analog zum Ereignismodell in Form von einfachen POJOs in Java umgesetzt werden. Anpassungen müssen nur für Planning Entities vorgenommen werden. Das bedeutet, dass bis auf die Klasse *ScheduleItem* alle anderen Planning Values ohne Anpassungen auskommen.
Die Klasse *ScheduleItem* wird mit Hilfe der *@PlanningEntity*-Annotation, als eben solches, Opta Planner bekannt gegeben. Außerdem wird das *item*-Attribut der Klasse *ScheduleItem* per Annotation als Planning Variable gekennzeichnet. Dabei wird dem Framework mitgeteilt, aus welchem Wertebereich die Planning Values zu wählen sind. In diesem Fall sind es die *OfferEvents*, die in der CGMU gespeichert sind.
Die vollständige Implementierung der ScheduleItems ist im Anhang im Listing D.1 hinterlegt.

Realisierung des Planungsvorgangs mit Jboss Opta Planner

Neben dem im vorangegangenem Abschnitt beschriebenen Modell der Planungsbestandteile bedarf es für die Realisierung des Planungsvorgangs einer konkreten Instanz des Planungsproblems. Opta Planner bietet hierzu das Interface *Solution<s extends Score>* an. Dieses stellt die eigentliche Repräsentation des Planungsproblems dar und stellt eine Methode zur Abfrage der ermittelten Bewertung einer Lösung bzw. eines Planungsschrittes bereit.

```
public class ProductionConsumptionBalancing implements
    Solution<HardAndSoftLongScore> {

   /*
    * Problem facts
    */

   // The grid entity offers
   private List<BaseOfferEvent> offers;
   // Placeholder events to represent "not used schedule items"
   private List<PlaceholderOfferEvent> dummies;
   // The total energy consumption in the grid
   // [Watt]
   private TotalEnergyConsumption totalElectricityConsumption;
   // The total energy production in the grid
   // [Watt]
   private TotalEnergyProduction totalElectricityProduction;

   ...

   /*
```

```
    * Problem entities
    */
   private List<ScheduleItem> schedule;

   @PlanningEntityCollectionProperty
   public List<ScheduleItem> getSchedule() {
      return schedule;
   }

   ...

   private HardAndSoftLongScore score;

   @Override
   public HardAndSoftLongScore getScore() {
      return this.score;
   }

   @Override
   public void setScore(HardAndSoftLongScore score) {
      this.score = score;
   }

   ...

}
```

Listing 8.3: Solution-Implementierung zur Beschreibung des Planungsproblems

Zur Problemformulierung wurde die Solution *ProductionConsumptionBalancing* erstellt. Sie repräsentiert das konkrete Planungsproblem des Anwendungsbereichs. Zusätzlich repräsentiert sie aber auch die verschiedenen Lösungsstadien und gefundenen Lösungen.
Für die letzten beide Punkte ist die Methode *cloneSolution()* des Solution-Interfaces zu implementieren, so dass bei einer neu gefundenen Lösung die Solution geklont werden kann und so die Lösung gesichert wird. Auf diese Weise kann auf eine bereits ermittelte Teillösung zurückgegriffen werden und so der Planungsvorgang, auf Kosten des Speicherverbrauchs, effizienter gestaltet werden.

Listing 8.3 zeigt einen Auszug aus der *ProductionConsumptionBalancing*-Solution. Anhand der Attribute kann man erkennen, dass es die Hauptaufgabe einer Solution ist, die aktuellen Planning Entities und Problem Facts vorzuhalten.
Die Planning Entities stellen eine Liste von *ScheduleItems* dar, die unter dem Attribut *schedule* hinterlegt werden. Die Anzahl an *ScheduleItems* richtet sich nach der Anzahl an *OfferEvents*. Da ein

OfferEvent maximal einmal genutzt werden kann, stellt die Anzahl an *offerEvents* gleichzeitig die Obergrenze für die Anzahl möglicher *ScheduleItems* dar.

Als Planning Facts werden zum Einen die aktuellen *OfferEvents* unter dem Attribut *offers* als Liste vorgehalten. Außerdem zählen die aktuellen Verbrauchs- und Produktionsmengen sowie eine Liste von Platzhalter-Elementen zu den Planning Facts.
Die aktuelle Menge an produzierter und verbrauchter Energie wird als Eingabe für die Bewertungsfunktion von Lösungen benötigt, da sich nur anhand des aktuellen Netzzustandes der Wert einer Lösung berechnen lässt. Hierzu wurden zwei Wrapper-Klassen, *TotalEnergieProduction* und *-Consumption*, erstellt. Die jeweils aktuellen Werte werden als Objekte dieser Klassen in den Attributen *totalElectricityConsumption* und *totalElectricityProduction* gespeichert.
Platzhalter- bzw. Null-Elemente von Typ *PlaceholderOfferEvent* werden aus zwei Gründen benötigt:

1. In manchen Situationen, wie z.B. einem bereits stabilen Netz, ist es am günstigsten keine Aktionen durchzuführen. *PlaceholderOfferEvents* bilden diesen Umstand nach, da sie eine bereits gute Bewertung nicht beeinflusst.

2. Opta Planner kann in der aktuellen stabilen Version nicht mit Null-Werten umgehen (zukünftige Versionen sollen dies unterstützen). Planning Variablen, die Null-Werte enthalten, sorgen dafür, dass ein Planning Entity als nicht initialisiert angesehen wird. Da manche Planungsalgorithmen eine initiale Lösung herbeiführen wollen bzw. sie voraussetzen, muss ein neutrales Element eingeführt werden welches sich von Null unterscheidet. Ansonsten kommt es zu Problemen während des Planungsprozesses.

Die Instanzen der Platzhalter-Elemente werden als Liste unter dem Attribut *dummies* in der Solution vorgehalten. Ihre Anzahl richtet sich nach der Anzahl an *ScheduleItems*, so dass immer auch die Lösung bei der keine Aktion durchgeführt wird, betrachtet werden kann.

Score-Berechnung

Um unterschiedliche Lösungen vergleichen zu können und ein Maß für die Güte einer Lösung zu haben, muss im Allgemeinen den Planungsalgorithmen eine Bewertungsfunktion übergeben werden. Diese ist vom betrachteten Problem und Anwendungsbereich abhängig und kann nicht automatisch generiert werden.
Jboss Opta Planner bietet auch beim Entwurf und der Umsetzung der Bewertungsfunktion Unterstützung an. Es werden folgende Möglichkeiten zur Score-Berechnung angeboten:

- Simple Java Score Calculation -> Implementierung des Interfaces *SimpleScoreCalculator<Sol extends Solution>*

- Imcremental Java Score Calculation -> Erben von der Basisklasse *AbstractIncrementalScoreCalculator<Sol extends Solution>*

- Drools Score Calculation -> Implementierung von Regeln zur Score-Berechnung als Drools Rule File

Die inkrementellen Methoden, zu denen auch die Berechnung mittels Drools-Regeln zählt, sind effizienter, da immer nur die Veränderung zur Score der vorangegangenen Lösung berechnet werden muss. In dieser Arbeit wurde auf die Score-Berechnung mittels Drools-Regeln zurückgegriffen. Hierzu soll zunächst im kommenden Abschnitt das generelle Vorgehen zur Score-Berechnung beschrieben werden. Danach wird die Umsetzung mit Drools vorgestellt.

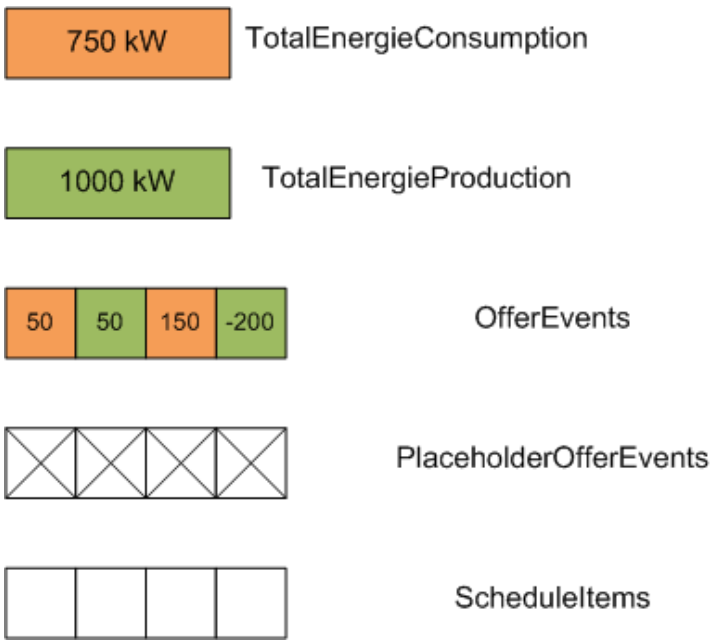

Abbildung 8.2.: Übersicht Bestandteile zur Score-Berechnung

Vorgehen zur Score-Berechnung

Allgemein bedarf es zur Berechnung eines Scores der folgenden Bestandteile (vgl. auch Abbildung 8.2):

- Aktueller Energieverbrauch
- Aktuelle Energieproduktion
- Offer- und PlaceholderOfferEvents
- Zu belegende ScheduleItems

Das Prinzip hinter der Score-Berechnung lautet wie folgt:

> Die aktuelle Energieproduktion und der aktuelle Energieverbrauch werden durch die in ScheduleItems genutzten OfferEvents erhöht oder verringert. Der Score ergibt sich aus der Differenz von Energieproduktion und -verbrauch. Bei einer optimalen Lösung ist die Differenz = 0.

Daraus ergeben sich die folgenden Formeln:

$$Energieverbrauch_{neu} = Gesamtenergieverbrauch + \{\text{alle verwendeten } ConsumptionOfferEvents\} \quad (8.1)$$

$$Energieproduktion_{neu} = Gesamtenergieproduktion + \{\text{alle verwendeten } ProductionOfferEvents\} \quad (8.2)$$

$$Score = |Energieverbrauch_{neu} - Energieproduktion_{neu}| \quad (8.3)$$

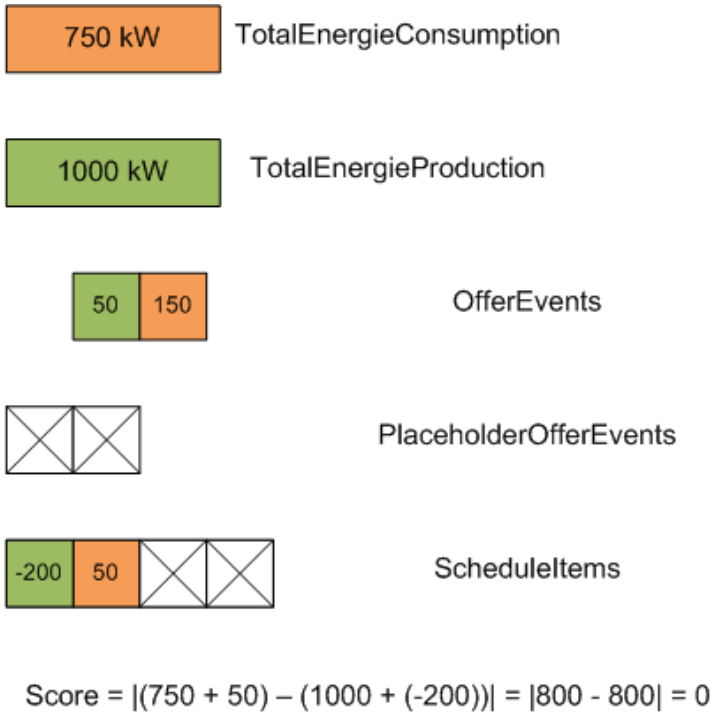

Abbildung 8.3.: Beispiel Score-Berechnung

Abbildung 8.3 verdeutlicht anhand eines konkreten Beispiels die Score-Berechnung. Hierbei wurden durch den Algorithmus das ProductionOfferEvent -200kW und das ConsumptionOfferEvent +50kW gewählt. Die restlichen ScheduleItems wurden mit Platzhaltern aufgefüllt, da bereits eine optimale Lösung gefunden wurde. Somit ergibt sich ein Energieverbrauch von 800kW und eine Energieproduktion von 800kW. Beide Größen sind ausgeglichen und die Differenz ist daher 0.

Hard- und SoftConstraints

Die Umsetzung dieses Vorgehens umfasst die Modellierung mithilfe von Hard- und SoftConstraints. HardConstraints stellen Bedingungen dar, die auf keinen Fall verletzt werden dürfen, damit es sich um eine gültige Lösung des Problems handelt. Im Gegensatz dazu stellen SoftConstraints Bedingungen dar, die nach Möglichkeit nicht verletzt werden sollen. Ihr Verletzung stellt aber auch nicht die Plausibilität der gefunden Lösung in Frage.

HardConstraints werden zur Modellierung folgender Bedingungen herangezogen:

1. Jedes *OfferEvent* darf nur einmal verwendet werden.
2. Pro *OfferEvent*-Kategorie (Consumption- oder ProductionOfferEvent) darf pro Netzteilnehmer nur ein Ereignis genutzt werden.

Die erste Bedingung wird automatisch von Opta Planner sichergestellt, da die Planning Facts jeweils nur einmal verwendet werden. Für die zweite Bedingung sind HardConstraints zu implementieren. Die Bedingung drückt dabei aus, dass sich die Angebots- und Nachfrageseite eines Netzteilnehmers unabhängig voneinander regeln lassen. Zwei Ereignisse der gleichen Seite des Netzteilnehmers schließen sich allerdings aus, da sie unterschiedliche Zustände der Seite repräsentieren und sich ein Netzteilnehmer nur in einem Zustand befinden kann. Konkret bedeutet das, wenn man den Verbrauch eines Teilnehmers um 50kW drosselt, darf kein weiteres Verbrauchsereignis in der Lösung vorkommen. Dieses würde das erste Verbrauchsereignis überschreiben und der tatsächliche Energieverbrauch würde nicht mehr mit dem über die Ereignisse geplanten übereinstimmen.
Um dies zu verhindern wurden folgende Regeln implementiert:

```
rule "onlyOneConsumptionOfferPerEntity"
   when
      $leftSchedItem : ScheduleItem( item != null, item.getClass() ==
          ConsumptionOfferEvent.class , $leftID : identifier, , $leftItemID :
          item.getEntityId() )
      $rightSchedItem : ScheduleItem( item != null, item.getClass() ==
          ConsumptionOfferEvent.class, identifier != $leftID, , item.getEntityId()
          == $leftItemID )
   then
       insertLogical(new
           LongConstraintOccurrence("onlyOneConsumptionOfferPerEntity",
           ConstraintType.NEGATIVE_HARD,
                 1,
               $leftSchedItem, $rightSchedItem));
    end

rule "onlyOneProductionOfferPerEntity"
```

```
   when
      $leftSchedItem : ScheduleItem( item != null, item.getClass() ==
          ProductionOfferEvent.class , $leftID : identifier, , $leftItemID :
          item.getEntityId() )
      $rightSchedItem : ScheduleItem( item != null, item.getClass() ==
          ProductionOfferEvent.class, identifier != $leftID, , item.getEntityId() ==
          $leftItemID )
   then
       insertLogical(new LongConstraintOccurrence("onlyOneProductionOfferPerEntity",
           ConstraintType.NEGATIVE_HARD,
                1,
                  $leftSchedItem, $rightSchedItem));
end
```

Listing 8.4: Regeln zur Überprüfung der zweiten Bedingung

Die erste Regel prüft, ob es für ein mit einem ConsumptionOfferEvent belegtes ScheduleItem ein anderes ScheduleItem gibt, welches auch ein ConsumptionOfferEvent beherbergt und vom gleichen Netzteilnehmer wie das erste, ist. Analog prüft die zweite Regel den Sachverhalt für ScheduleItems mit ProductionOfferEvents.
Wann immer zwei ScheduleItems gefunden werden, die zusammen eine der beiden Regeln erfüllen, wird ein neues *ConstraintOccurence*-Objekt erzeugt, dass die Verletzung eines negativen HardConstraints signalisiert. Mit Hilfe der Regel in Listing 8.5 wird nach jedem Planungsschritt die Anzahl an HardConstraint-Verletzungen ermittelt und als Bestandteil der Score an die Solution weitergegeben. Ausschlaggebend dafür, dass alle Constraint-Verletzungen erfasst werden können, ist, dass die Regel einen *salience*-Wert von -1 hat und somit erst nach allen in Listing 8.4 beschriebenen Regeln zur Ausführung kommt. Auf diesem Weg können dann alle im Working Memory der Rule Engine vorhandenen *ConstraintOccurrence*-Objekte erkannt werden.

```
   rule "hardConstraintsBroken"
        salience -1
    when
        $hardTotal : Number() from accumulate(
            LongConstraintOccurrence(constraintType == ConstraintType.NEGATIVE_HARD,
                $weight : weight),
            sum($weight)
        )
    then
        scoreHolder.setHardConstraintsBroken($hardTotal.longValue());
end
```

Listing 8.5: Regeln zur Ermittlung der Anzahl an HardConstraint-Verletzungen

SoftConstraints werden zur Modellierung der Differenz zwischen produzierter und verbrauchter Energie genutzt. Es handelt sich dabei um einen negativen SoftConstraint, da die Differenz zu minimieren ist. Im Idealfall soll sie 0 betragen.

```
rule "Get initial production value"

   when
      $totalProduction : TotalEnergyProduction( $value : productionValue )
    then
      insertLogical(new ProductionValue($value));

end

rule "Get initial consumption value"

   when
      $totalConsumption : TotalEnergyConsumption( $value : consumptionValue )
   then
      insertLogical(new ConsumptionValue($value));

end

rule "Create new production value when offer event is used"

   when
      $productionScheduleItem : ScheduleItem( item != null, item.getClass() ==
          ProductionOfferEvent.class, $changeValue : item.changeValue )
   then
       insertLogical(new ProductionValue($changeValue));
end

rule "Create new consumption value when offer event is used"

   when
      $consumptionScheduleItem : ScheduleItem( item != null, item.getClass() ==
          ConsumptionOfferEvent.class, $changeValue : item.changeValue )
   then
       insertLogical(new ConsumptionValue($changeValue));
end
```

Listing 8.6: Regeln zur Erzeugung von ProductionValue- und ConsumptionValue-Objekten

Abgebildet wird dieser Mechanismus über die in Listing 8.6 dargestellten Regeln. Wird ein *TotalEnergieProduction*- oder *TotalEnergieConsumption*-Objekt im Working Memory entdeckt, so wird dessen

Wert extrahiert und ein korrespondierendes ProductionValue- oder ConsumptionValue-Objekt mit dem Wert erzeugt.
Das Gleiche passiert auch, sobald ein ScheduleItem mit einem neuen OfferEvent belegt wurde. Hier wird der Wert des Attributs *changeValue* extrahiert und ein passendes Value-Objekt erzeugt.

```
rule "Calculate simple score"
   salience -1
   when
      $futureProduction : Number() from accumulate (
         ProductionValue($production : value),
         sum($production)
      )
      $futureConsumption : Number() from accumulate (
         ConsumptionValue($consumption : value),
         sum($consumption)
      )
   then
      // Create negative score out of the production consumption difference
      long diff = Math.abs($futureProduction.longValue() -
          $futureConsumption.longValue());
      long score = diff;
      scoreHolder.setSoftConstraintsBroken(score);
end
```

Listing 8.7: Regel zur abschließenden Ermittlung der Differenz zwischen Produktions- und Verbrauchswerten

Abschließend werden alle ProductionValue- und ConsumptionValue-Objekte separat aufsummiert und auf diesem Weg die Differenz zwischen den zukünftigen Produktions- und Verbrauchswerten ermittelt (vgl. Listing 8.7). Die ermittelte Differenz wird als SoftConstraint-Score an die Solution übergeben.

8.2.4. Zusammenfassung

Die Steuerung des Stromnetzes konnte durch den Einsatz von *Jboss Drools* zur Ereignisverarbeitung und *Jboss Opta Planner* zur automatisierten Lösung von Planungsproblemen realisiert werden.
Hierzu musste zuerst das zu lösende Planungsproblem identifiziert und formal beschrieben werden. Im Anschluss erfolgte die Modellierung des Problems mit Hilfe der durch Opta Planner bereitgestellten Hilfsmittel.
Auch wenn das Übereinbringen von Hilfsmittel und Problembestandteilen auf den ersten Blick nicht immer ersichtlich ist, zahlt sich der zusätzliche Aufwand im späteren Verlauf aus. Der Aufwand für die Durchführung der anschließenden Untersuchungen zur Bestimmung eines geeigneten Pla-

nungsalgorithmus kann, auf Grund der bereits implementierten Algorithmen und des vorhanden Benchmark-Werkzeugs, deutlich verringert werden.

8.3. Simulationsumgebung

Die Oberfläche der Simulationsumgebung wurde im Hinblick auf die spätere Touch-Bedienung hin optimiert. Das Kernkonzept stellen dabei Drag&Drop-Gesten zum Hinzufügen und Entfernen von Netzteilnehmern dar. Hiermit konnte der nicht-funktionalen Anforderung *NF5* in Abschnitt 5.4.2 Rechnung getragen werden.

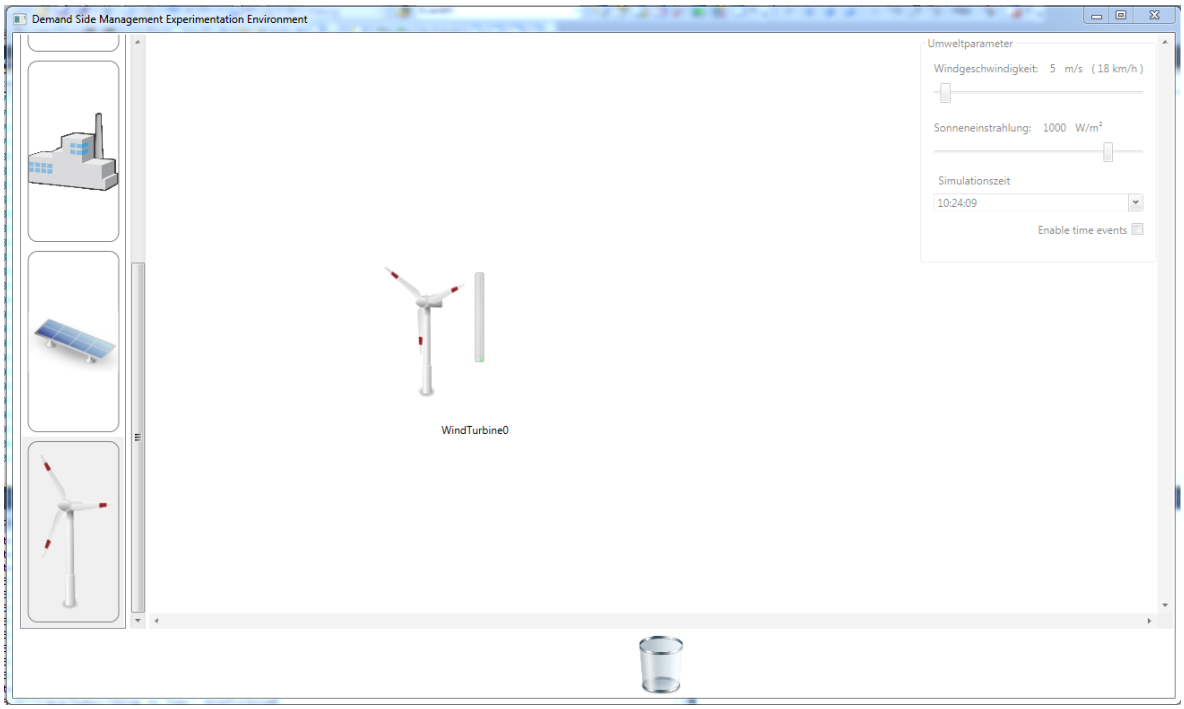

Abbildung 8.4.: Oberfläche der Simulationsumgebung

In Abbildung 8.4 ist die Oberfläche zur Steuerung der Simulationsumgebung abgebildet. Sie ermöglicht es, das Simulationsszenario dynamisch in Echtzeit zu verändern, wohingegen der geläufige Ansatz darin besteht, ein Szenario im Vorfeld zu modellieren und dieses dann über einen festen Zeitraum zu simulieren (vgl. andere Ansätze aus dem Kapitel 3).
Über die Netzteilnehmerleiste am linken Bildrand können neue Netzteilnehmer der Simulation hinzugefügt werden, indem ein Icon per Drag&Drop auf die angrenzende Simulationsfläche gezogen wird. Das Entfernen von Netzteilnehmern geschieht über das Mülleimer-Symbol im unteren Bildrand. Hierzu werden die Netzteilnehmer-Symbole, auch wieder per Drag&Drop, in den Mülleimer gezogen. Zur Steuerung der Umweltfaktoren steht in der rechten oberen Ecke ein Panel zur Verfügung. Die Werte können mithilfe der Slider dynamisch, in den gegeben physikalischen Grenzen, verändert werden. Hierdurch können die Simulation und die Netzteilnehmer mittelbar beeinflusst werden.

8.3.1. Softwaretechnische Umsetzung

Bei der Umsetzung wurde auf der in Abschnitt 6.1 beschriebenen Architektur aufgebaut. Kombiniert wurde die vorgesehene Architektur mit dem *Model-View-ViewModel-Entwurfsmuster*[12] (MVVM), welches im .NET-Umfeld den Standard zur Verbindung von Backend und Benutzeroberfläche darstellt.

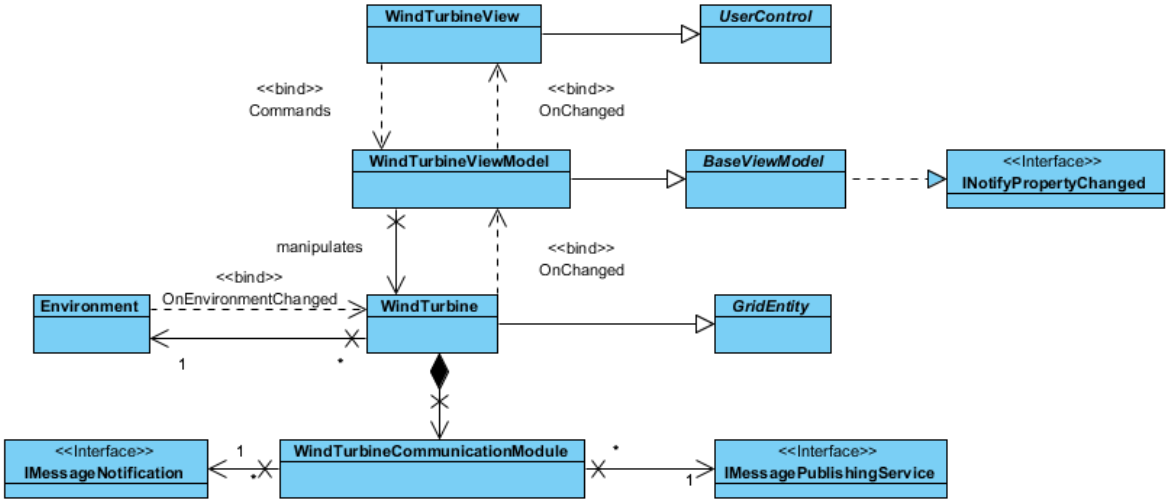

Abbildung 8.5.: MVVM-Hierarchie anhand eines Windrades

Im Rahmen der Kombination des Softwarearchitekturentwurfs und des MVVM-Musters entstand die in Abbildung 8.5 dargestellte Klassenhierarchie. Zwischen dem ursprünglichen Architekturentwurf (vgl. Abbildung 6.1) und der in Abbildung 8.5 dargestellten Umsetzung wurden zwei Anpassungen vorgenommen.
So wurden die Komponenten *ControlLogic* und *GridEntityModel* zu einer Model-Komponente verschmolzen (in Abbildung 8.5 durch die Klasse *WindTurbine* dargestellt). Hierdurch konnte eine Ebene in der Klassenhierarchie eingespart werden. Und da die Steuerungslogik über einen endlichen Automaten realisiert wurde, kann dieser auch direkt in der Modell-Instanz erzeugt werden, da das eingesetzte Framework die Austauschbarkeit der Implementierung gewährleistet.
Zusätzlich wurde zum Entwurf eine Klasse für das *ViewModel* (hier *WindTurbineViewModel*) eingeführt. Sie bündelt die Validierungs- und andere Logik zur Präsentation.
Mit Hilfe der Softwarearchitektur, des Einsatzes des MVVM-Musters und der integrierten Data-Binding-Fähigkeiten von C# konnten Backend, Netzteilnehmermodell und -repräsentation so verbunden werden, dass Änderungen in der Simulation in Echtzeit auf der Benutzeroberfläche angezeigt werden.

[12] `http://msdn.microsoft.com/de-de/magazine/dd419663.aspx`

8.4. Monitoringumgebung

Bei der Entwicklung des Monitoring-Teilsystems mussten im Gegensatz zur Simulationsumgebung keine Anpassungen an der Softwarearchitektur vorgenommen werden. Die in Abschnitt 6.2 beschriebene und in Abbildung 6.2 dargestellte Komponentenstruktur wurde in JavaScript eins zu eins umgesetzt.
Zur Kommunikation mit dem Event Store (einer CouchDB-Instanz) wurde das durch CouchDB mitgelieferte Plugin *$.couch*[13] für jQuery genutzt. Es bietet Methoden an, die die Ajax-Aufrufe für die CouchDB-Funktionen kapseln und so die Interaktion mit CouchDB deutlich vereinfachen. Auf diese Weise konnte die Change-API dazu genutzt werden, das Monitoring über neu ankommende Ereignisse zu informieren (vgl. die in Listing 8.8 dargestellte Funktionalität).

```
...
// Subscribe for changes in database
   $.couch.db(dbname).changes().onChange(function(data) {

      console.log(data);

      // Get the new document
      $.couch.db(dbname).openDoc(id, {
         success : function(data) {
            // debugger;
            var eventType = eventUtils.getEventType(data);

            console.log("Received event of type: " + eventType);

            switch(eventType) {
               case joinEvent:
                  console.log("Received join event.");
                  // Notify view model about event
                  gridEntitiesViewModel.handleJoinEvent(data);
                  break;
               case leaveEvent:
                  console.log("Received leave event.");
                  // Notify view model about event
                  gridEntitiesViewModel.handleLeaveEvent(data);
                  break;
               case productionChangedEvent:
                  console.log("Received production changed event.");
                  // Handle event
                  gridStatusController.trigger("change:production", data);
```

[13] http://daleharvey.github.io/jquery.couch.js-docs/symbols/index.html

```
                break;
            case consumptionChangedEvent:
                console.log("Received consumption changed event.");
                // Handle event
                gridStatusController.trigger("change:consumption", data);
                break;
            case frequencyChangedEvent:
                console.log("Received frequency changed event.");
                // Handle event
                gridStatusController.trigger("changed:frequency", data);
                break;
            default:
                console.log("Received unimportant event or no event.");
        }

      },
      error : function(status) {
         console.log(status);
      }
   });

});
...
```

Listing 8.8: Ereignismechanismus für CouchDB

Durch den Einsatz des Frameworks *Backbone.JS* konnte eine ereignisgesteuerte Kommunikation der Komponenten erreicht werden. Model-, View- und Controller-Instanzen nutzen dazu den Eventbus, der durch *Backbone.JS* bereitgestellt wird. So konnte eine vollständig ereignisgesteuerte Monitoringumgebung entwickelt werden, die in Abbildung 8.6 dargestellt ist.

8.4.1. Benutzungsoberfläche

Die Benutzeroberfläche besteht aus einem Graphen, der den zeitlichen Verlauf der Frequenz widerspiegelt und einem Graphen für den zeitlichen Verlauf der Verbrauchs- und Produktionswerte. Außerdem existiert eine alternative Darstellung der aktuellen Netzfrequenz in Form eines Tachometers. Diese Darstellungsform ermöglicht es, auf einen Blick die Abweichung von der Normfrequenz zu erfassen.
Realisiert wurden die Anzeigen mit der Bibliothek *highcharts*[14]. Sie stellt unterschiedliche Typen von Charts zur Verfügung, die sich durch die mitgelieferten Funktionen leicht an den jeweiligen Verwendungszweck anpassen lassen.

[14] http://www.highcharts.com/

Abbildung 8.6.: Benutzeroberfläche des Monitoring

Besonders interessant für eine Monitoringlösung ist die von der Bibliothek angebotene Exportfunktion. Über einen durch *highchart* bereitgestellten Server, oder wahlweise auch einem eigenen Server, können die Graphen als pdf, png, jpeg oder als svg Vektorgrafik exportiert und gespeichert werden.

8.5. Zusammenfassung

Das Ergebnis der Realisierungsphase ist ein über weite Teile hinweg ereignisgesteuertes System. Die genutzten Frameworks und Bibliotheken erlauben lose gekoppelte Teilsysteme, welche in unterschiedlichen Programmiersprachen und -plattformen erstellt wurden, zu einem Gesamtsystem zu verbinden. Die zentralen Bindeglieder sind dabei das Ereignismodell und das verwendete Transportformat JSON.
Durch den Einsatz von *Jboss Opta Planner* konnte der Planungsprozess und die zugrundeliegende Problemstellung in Java umgesetzt werden. Auf diese Weise kann das Stromnetz durch die CGMU automatisch gesteuert werden. Die in diesem Kapitel vorgestellte Realisierung bildet zusätzlich die Grundlage für die in Kapitel 10 durchzuführende Evaluierung von Planungsalgorithmen. Da in diesem Kapitel nur die programmatische Realisierung des Planungsprozesses erläutert wurde, ist die Netzsteuerung erst im eigentlichem Sinne einsatzbereit, nachdem ein oder mehrere geeignete Planungsalgorithmen gefunden wurden. Diese Betrachtung erfolgt in Kapitel 10.

9. Test

Um die Funktionalität der entwickelten Systeme und ihrer Komponenten zu verifizieren, wurden Tests auf verschiedenen Ebenen durchgeführt. Dabei wurde generell nach Prinzip des *Test Driven Developement* (TDD) vorgegangen. Dieses Vorgehen hatte sich schon in Steudte (2011) bewährt und stellt sicher, dass trotz der lose gekoppelten Teilsysteme und Komponenten Fehler frühzeitig erkannt und die geforderte Funktionalität verifiziert werden kann.

9.1. Allgemeine Grundsätze

Die Softwaretests konzentrieren sich auf die Kernfunktionalitäten und die zentralen Schnittstellen. Für im Entwicklungsprozess auftretende Fehler wurden zusätzliche Tests zur Lokalisierung der Fehlerquelle geschrieben. Auf Grund des Umfangs und der Anzahl an Teilsystemen wurden im Vorfeld der Implementierung nur Tests für die angesprochenen Kernfunktionalitäten entwickelt, um sich im zur Verfügung stehenden zeitlichen Rahmen der Arbeit auf die Umsetzung der Funktionalität konzentrieren zu können. Bei einer strikten Einhaltung des TDD-Ansatzes hätte, bevor nicht ein Test geschrieben wurde, keine Implementierung der Funktionalität erfolgen dürfen. So wurden kleine und überschaubare Funktionseinheiten hiervon ausgenommen, um zusätzlichen zeitlichen Spielraum zu gewinnen.

9.2. Vorgehen

Wie im vorangegangenen Abschnitt bereits erwähnt wurde, wurden zuerst für die zentralen Funktionen der zu entwickelnden Komponente die Komponententests entworfen und geschrieben. Anschließend erfolgte die Implementierung der Funktionalität, so dass die Tests nicht mehr fehlschlugen.
Damit wurde aber nur die unterste Ebene der Testhierarchie betrachtet, die in der folgenden Aufzählung dargestellt ist:

1. Komponententest
2. Isolierter Test der Schnittstellenkomponenten zu anderen Teilsystemen
3. Integrationstest auf Komponentenebene
4. Integrationstest auf Systemebene

Wenn es sich bei der zu entwickelnden Komponente um eine Komponente handelte, die eine Verbindung zu einem anderen Teilsystem herstellt, wurde aufbauend auf den Komponententests soweit möglich ein Integrationstest zwischen der Komponente und den beteiligten Teilsystemen vorgenommen. Durch diesen Ansatz konnten frühzeitig Fehler im Zusammenspiel der Teilsysteme erkannt und beseitigt werden. Die durch die lose Kopplung entstehenden Schwierigkeiten lassen sich so abmildern.
Spätestens nach der Fertigstellung aller Komponenten eines Teilsystems erfolgte der Integrationstest auf Komponentenebene. Durch ihn wird sichergestellt, dass die Komponenten korrekt zusammenarbeiten.
Nach der Implementierung einzelner Teilsysteme wurden die Integrationstests auf der Systemebene durchgeführt und so das Zusammenspiel der Teilsysteme überprüft.

9.3. Eingesetzte Hilfsmittel

Bei der Durchführung von Softwaretests steht zumeist die Zustandsverifikation im Vordergrund. Neben der Verifikation des Zustands einer Komponente kann auch das Verhalten einer Komponente mittels Tests betrachtet werden. Die Verhaltensverifikation hat sich als äußerst nützlich zur Überprüfung der Reaktion auf Ereignisse erwiesen.
Für die Zustandsverifikation wurde unter Java auf das Framework JUnit[1] 4 zurückgegriffen. Auf Seiten von .NET wurde das in Visual Studio 2010 integrierte Testframework für diese Aufgabe verwendet.
Zur Verhaltensverifikation wurden *moq*[2] und *EasyMock*[3] verwendet. Die beiden Frameworks wurden auch dazu genutzt, Komponenten und Teilsysteme isoliert testen zu können. Zu diesem Zweck wurden die benötigten Abhängigkeiten als Stub- oder „nice“ Mock-Objekt nachgebaut. Wo keine Verhaltensverifikation benötigt wird, wurde auf ihren Einsatz verzichtet. Stattdessen wurden Stub-Objekte verwendet. Das Listing 9.1 zeigt ein Beispiel für die Erstellung von Stub-Objekten in Java durch *EasyMock*.

```
...

   @Before
   public void setUp() throws Exception {

      IEventNotification msgEventTransformer =
          EasyMock.createNiceMock(IEventNotification.class);
      IEventMessageTransformer eventMsgTransformer =
          EasyMock.createNiceMock(IEventMessageTransformer.class);
```

[1] http://junit.org/
[2] https://code.google.com/p/moq/
[3] http://easymock.org/

```
      EasyMock.replay(msgEventTransformer);
      EasyMock.replay(eventMsgTransformer);

      cgmu = new CentralGridManagementUnit(eventMsgTransformer,
          msgEventTransformer);
      cgmu.init();

   }

...
```

Listing 9.1: Erzeugung von Stub-Objekten für Ereigniskomponenten

9.4. Verifikation der Planungs- und Ereiegnisfunktionalität

Um die Ergebnisse der Planungs- und Ereignisverarbeitung überprüfen zu können, ist die Verhaltensverifikation von großem Nutzen. Durch den Einsatz von Mock-Objekten, und deren Fähigkeit zur Evaluierung von Methodenaufrufen, lassen sich auch asynchrone Methodenaufrufe, Callbacks sowie die bei den Aufrufen verwendeten Parameter überprüfen (vgl. Listing 9.2).

```
...
// Expect subscribing to and unsubscribing from msgEventTransformer
EasyMock.expect(msgEventTransformer.subscribeNotifyee(
   EasyMock.and(EasyMock.notNull(CentralGridManagementUnit.class),
       EasyMock.isA(CentralGridManagementUnit.class))))
   .andReturn(true);
EasyMock.expect(msgEventTransformer.unsubscribeNotifyee(
   EasyMock.and(EasyMock.notNull(CentralGridManagementUnit.class),
       EasyMock.isA(CentralGridManagementUnit.class))))
   .andReturn(true);

// Expect callback event for full-throttling to be send
eventMsgTransformer.SubmitEventTo(entity, fullThrottle);

EasyMock.replay(msgEventTransformer, eventMsgTransformer);
...
```

Listing 9.2: Callback-Verifizierung durch Mock-Objekte

Sowohl in Java als auch in C# erschweren asynchrone Aufrufe die Verifikation von Objektzuständen. Es kann vorkommen, dass das Ergebnis eines asynchronen Aufrufes erst nach dem Aufruf der überprüfenden Assertion zur Verfügung steht. In einfachen Fällen kann eine Sleep-Anweisung zum Warten auf das Ergebnis genutzt werden. Bei komplexeren Fällen nutzt man besser die Verhaltensve-

rifikation für Events bzw. Callback-Methoden. Auf diese Weise können auch komplexe Aufruf und deren Ergebnisse geprüft werden.
Ein Beispiel hierfür ist im Anhang unter Listing E.1 hinterlegt. In dem Beispiel wird geprüft, ob der MessageAdapter der Simulationsumgebung Nachrichten über die Middleware empfangen kann, indem er eine Nachricht an sich selber versendet. Über ein *AutoResetEvent* wird auf den Empfang der Nachricht mindestens eine Sekunde gewartet und erst im Anschluss die Assertion ausgeführt, die überprüft ob die Nachricht empfangen wurde.

Um die Ereignisverarbeitung und den Planungsprozesses überprüfen zu können, wurden unterschiedliche Szenarien entwickelt. Einige von ihnen werden in Abschnitt 10.2 im Rahmen der Untersuchung von Planungsalgorithmen genauer beschreiben. Aus diesem Grund soll an dieser Stelle auf eine Beschreibung verzichtet werden.
Generell werden dabei die entworfenen Szenarien in eine Folge von Ereignissen übersetzt. Diese Ereignisse wurden mithilfe der in den Szenarien beschriebenen Testdaten erstellt und dann der Middleware oder direkt der zu testenden Komponenten übergeben. Auf diesem Weg wird die Interaktion der Teilsysteme auf Ebene der Ereignisse nachgeahmt und die entwickelte Funktionalität getestet.

9.5. Erfahrungen

Die Durchführung der Tests erfolgte manuell in den Entwicklungsumgebungen. Der initiale Aufwand eine Umgebung für *Continous Integration*(CI), wie z.B. Jenkins[4], einzurichten wäre auf Grund des Einsatzes mehrerer Entwicklungsplattformen zu aufwändig gewesen. Im Nachhinein stellte sich heraus, dass auch Plugins zur Integration von MSBuild in Jenkins existieren und so mehrere Plattformen ohne Problem integriert werden können. Außerdem hatte zum Ende der Entwicklung das Projekt eine Größe erreicht, bei der die automatische Durchführung von Builds und Tests sehr vorteilhaft ist. Bei einem weiteren Projekt dieser Größenordnung sollte daher der initiale Aufwand des Aufsetzens einer CI-Umgebung in Kauf genommen werden. Über die gesamte Projektlaufzeit betrachtet, rentiert sich der einmalige Aufwand für die Einrichtung.
Weitere Erkenntnisse konnten bezogen auf den Umgang mit Ereignissen und asynchronen Funktionsaufrufen gewonnen werden. Wie bereits im Abschnitt 9.4 berichtet wurde, werden zu deren Verifikation andere Ansätze und Techniken benötigt.

9.6. Zusammenfassung

Die lose Koppelung der Teilsysteme und der Einsatz von asynchronen Funktionsaufrufen stellen neue Herausforderungen daran, wie Softwaretests zu schreiben sind. Durch ein testgetriebenes Vorgehen und mehrere Testebenen konnten Fehler frühzeitig erkannt und behoben werden. Vor allem Fehler

[4] `http://jenkins-ci.org/`

in der Interaktion der Teilsysteme konnten so effizient beseitigt werden. Ohne ein solches Vorgehen wäre der Aufwand für die Entwicklung des beschriebenen Systems deutlich größer gewesen.
Als besonders hilfreich hat sich der Einsatz von Verhaltensverifikation, neben der bekannten Zustandsverifikation, in Verbindung mit der Überprüfung von Callback-Methoden und C# Events erwiesen. Mit Hilfe von Mock-Objekten konnten Fehler bei der Ereignisverarbeitung und im Planungsprozess mit wenig Aufwand ermittelt und beseitigt werden.
In der Nachschau wäre der Einsatz einer CI-Lösung zur automatisierten Testdurchführung sinnvoll gewesen. Da für ein System dieser Größe viele Tests nötig sind und deren manuelle Ausführung einige Zeit in Anspruch nimmt, ließe sich der initiale Aufwand über die gesparte Entwicklungszeit kompensieren. Auch könnte man so Fehler, die durch Änderungen entstanden sind, automatisch entdecken. Für zukünftige Projekte wird auf Grund dieser Erfahrung die Bereitstellung und Einrichtung einer CI-Umgebung ausdrücklich empfohlen.

10. Untersuchung der Planungsalgorithmen

Die in Kapitel 8 vorgestellte Realisierung des Planungsproblems ist noch nicht vollständig. Neben der bereits besprochenen Modellierung und der darauf aufbauenden Realisierung mittels Opta Planner, sind auch Untersuchungen bzgl. geeigneter Planungsalgorithmen nötig. Daher sollen in diesem Kapitel die verschiedenen Algorithmen verglichen werden.
Hierzu werden zuerst Szenarien definiert. Anschließend werden sie durch die betrachteten Algorithmen einzeln gelöst und die Ergebnisse protokolliert. Die gesammelten Ergebnisse werden im Anschluss gegenübergestellt, um so einen geeigneten Algorithmus zu ermitteln.

10.1. Algorithmen

Bei Planungsalgorithmen unterscheidet man zwischen zwei Arten von Algorithmen:

1. Construction Heuristic
2. Local Search (Metaheuristic)

Algorithmen aus dem Bereich der *Construction Heuristic* ermitteln in endlicher Zeit eine initiale Lösung. Die gefundene Lösung kann dann durch *Local Search* zu immer besseren Lösungen verfeinert werden.
Für beide Kategorien werden durch Opta Planner bereits implementierte und getestete Algorithmen angeboten. Vertreter aus dem Bereich der Construction Heuristic sollen im Folgenden einzeln vorgestellt werden. Es soll darüber hinaus jeweils geklärt werden, ob der jeweilige Algorithmus überhaupt angewendet werden kann und ob seine Anwendung sinnvoll ist.

Betrachtete Algorithmen

Bei den Untersuchungen wurde sich auf die Algorithmen *First Fit* und *Best Fit* konzentriert. *First Fit* initialisiert die Planning Entities Schritt für Schritt mit dem besten verfügbaren Wert. Wohingegen *Best Fit* auf eine Sortierung der Planungswerte aufbaut und die schwächeren Werte (hier OfferEvents mit einem großen Change Value) zuerst nutzt.

Nicht betrachtete Algorithmen

Nicht betrachtet wurden die Algorithmen *First Fit Decreasing* und *Best Fit Decreasing*. Bei beiden Algorithmen handelt es sich um Erweiterungen der ursprünglichen Algorithmen *First Fit* und *Best Fit*.

Die Decreasing-Varianten der beiden Algorithmen fallen aus der Betrachtung heraus, weil sie an die Planning Entities die Anforderung stellen, dass diese sich nach Schwierigkeit der Zuweisung sortieren lassen. In beiden Fällen würde versucht werden den schwieriger zu belegenden Planning Entities zuerst einen Wert zuzuweisen.
Bei dem hier vorliegenden Planungsproblem sind die ScheduleItems (vgl. Abschnitt 7.2.2) die Planning Entities. Da alle ScheduleItems gleich schwierig zu belegen sind, abgesehen von schon bereits verplanten ScheduleItems (dieses wird von den Algorithmen implizit berücksichtigt), ist der Einsatz von *First Fit Decrease* und *Best Fit Decrease* nicht sinnvoll. Die beiden Algorithmen würden sich wie ihre Ausgangsvarianten verhalten.

10.2. Szenarien

Nach der Formulierung des Planungsmodells und der Umsetzung mithilfe von *Jboss Opta Planner* stand im nächsten Schritt die Evaluierung der Planungsalgorithmen an, um die oder den geeigneten Algorithmus zu ermitteln. Hierzu wurden Szenarien definiert, welche verschiedene Anwendungsfälle aus dem Anwendungsbereich nachstellen. Hierbei standen sowohl alltägliche Szenarien als auch Grenzfälle im Mittelpunkt der Betrachtungen.
Die Szenarien wurden auch dazu genutzt, um die CGMU und ihre Planungskomponente isoliert von den restlichen Teilsystemen zu testen. So wurden die Szenarien grundsätzlich so realisiert, dass die Abläufe des Netzes über die Übergabe von Ereignissen an die CGMU simuliert wurde.
Im weiteren Verlauf dieses Abschnitts sollen die Szenarien, sowie die dabei verfolgte Zielsetzung, vorgestellt und erläutert werden. Die Implementierung der Szenarien ist im Anhang unter Listing C.1 hinterlegt.

10.2.1. Szenario 1

Szenario 1 stellt einen simplen Test der Planungsfähigkeit der CGMU dar. Um die Produktionsmenge von 1000kW und die Verbrauchsmenge von 750kW anzugleichen, soll die CGMU die Lösung [[COEV +50],[POEV -200]] ermitteln. „COEV“ steht in dieser Notation für ConsumptionOfferEvent und „POEV“ für ProductionOfferEvent. Abbildung 10.1 veranschaulicht das Szenario visuell.

10.2.2. Szenario 2

Auch dieses Szenario startet mit einem Gesamtverbrauch von 750kW und einer Energieproduktion von 1000kW. Das Szenario lässt sich durch den Einsatz von *First Fit* nicht optimal lösen. Durch die Wahl von [COEV 300] (erster den Score verbessernder Schritt) werden von der Construction Heuristic alle anderen Zweige des Lösungsbaums nicht mehr betrachtet. Dadurch kann die optimale Lösung [[COEV -50],[POEV -300]] nicht mehr erreicht werden. Aus diesem Grund muss die Lösung des Szenarios [[COEV 300]] lauten, wenn die eingesetzten Regeln nicht einen minimalen Verbrauch erreichen wollen.

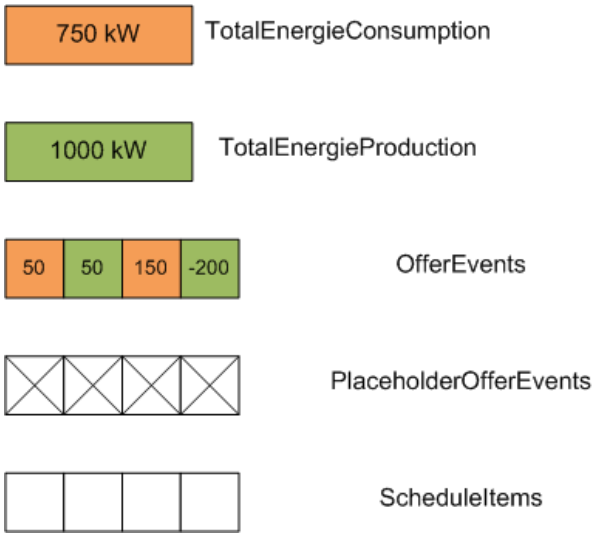

Abbildung 10.1.: Testszenario 1

Außerdem kann mit diesem Szenario auch getestet werden, ob die in Abschnitt 8.2.3 beschriebene Bedingung, bezüglich mehrerer OfferEvents von einem Typ pro Netzteilnehmer, durch die HardConstraints umgesetzt werden. Ohne die Bedingung wäre auch [[COEV 300],[COEV -50]] eine gültige Lösung. Die grafische Beschreibung befindet sich in Abbildung 10.2.

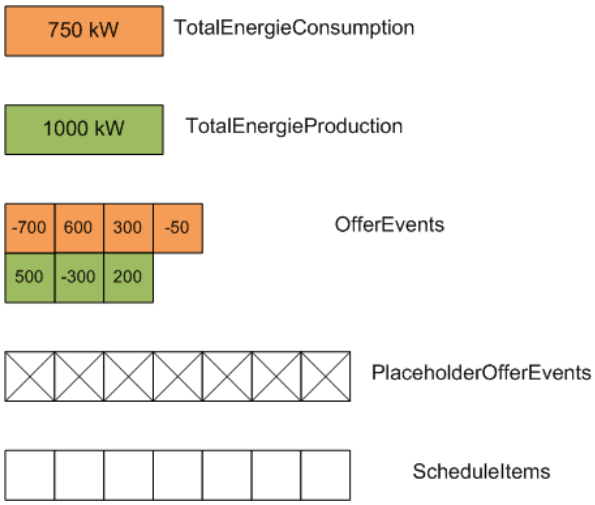

Abbildung 10.2.: Testszenario 2

10.2.3. Szenario 3

Hier wird geprüft wie sich das System verhält, wenn nur ProductionOfferEvents verfügbar sind und die Menge an ConsumptionOfferEvents leer ist (vgl. Abbildung 10.3).

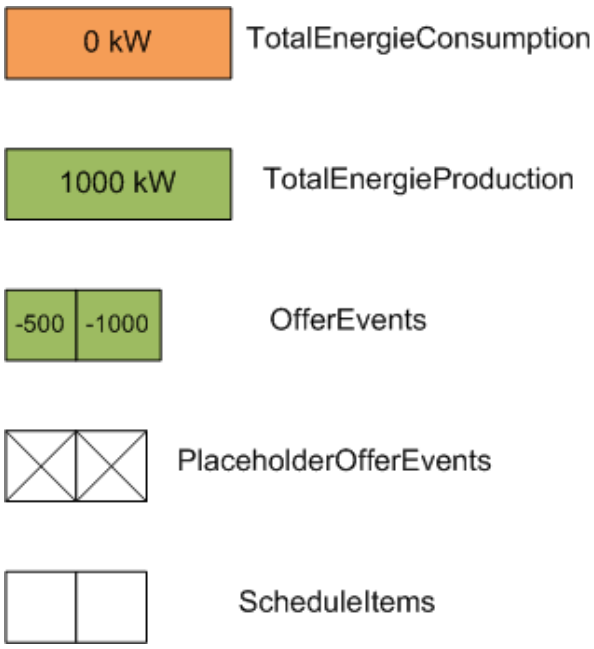

Abbildung 10.3.: Testszenario 3

10.2.4. Szenario 4

Prüft die Umkehrung des Szenarios 3. In diesem Fall liegen nur ConsumptionOfferEvents vor und die Menge an ProductionOfferEvents ist leer (vgl. Abbildung 10.4).

10.2.5. Szenario 5

Dieses Szenario ist auf Grund der Anzahl der Ereignisse deutlich komplexer und dient damit als Lasttest. Es ist in etwa viermal komplexer als die anderen Szenarien. Die zu findende Lösung lautet [[POEV -300],[COEV -50]]. Die grafische Beschreibung befindet sich in Abbildung 10.5.

10.2.6. Szenario leeres Balancing-Problem

Hier wird der äußerste Grenzfall der gerade gestarteten Simulationsumgebung betrachtet. Da noch keine Netzteilnehmer existieren, sind somit die Mengen der OfferEvents leer. Aus diesem Grund existieren auch keine ScheduleItems und PlaceholderOfferEvents. Die dazugehörigen Mengen sind also auch leer. Produktion und Verbrauch liegen beide bei 0kW. Als Ergebnis wird erwartet, dass der Planungsvorgang ohne Ergebnis abgebrochen wird. Dabei soll es aber nicht zu Abstürzen oder Exceptions kommen.

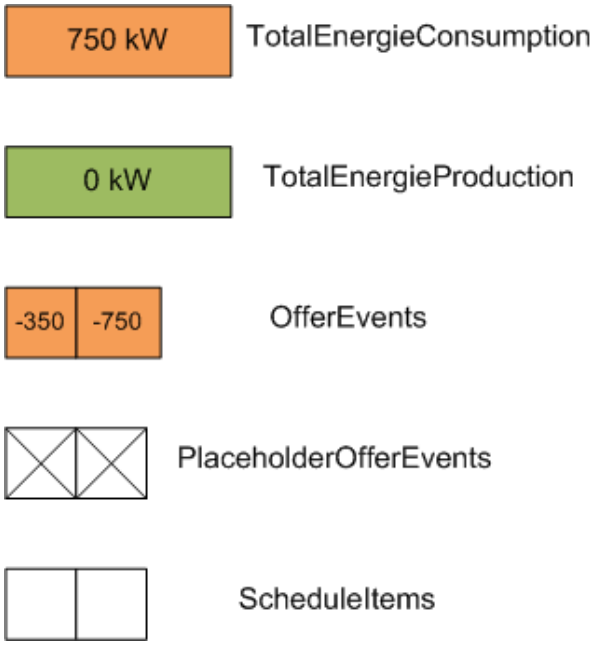

Abbildung 10.4.: Testszenario 4

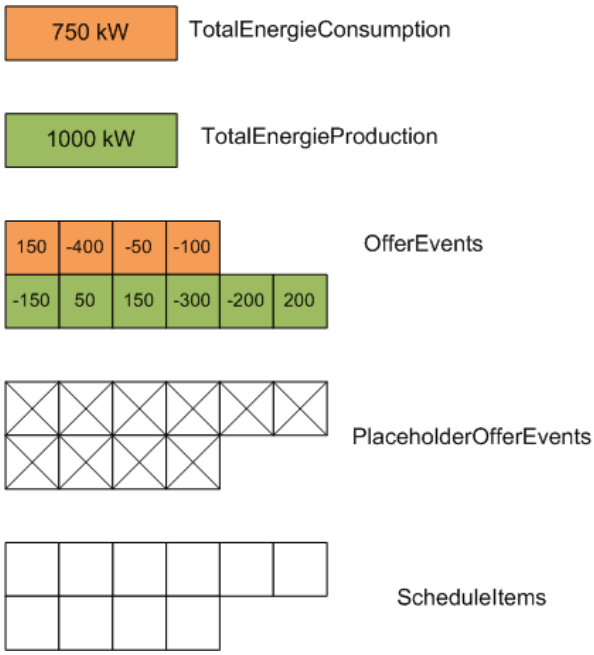

Abbildung 10.5.: Testszenario 5

10.3. Messungen und Ergebnisse

10.3.1. Ziele

Primäres Ziel der Untersuchungen in diesem Abschnitt ist es, einen Vergleich zwischen den Algorithmen *First Fit* und *Best Fit* zu ziehen. Hierbei steht nicht nur das Ermitteln der für jedes Szenario definierten Lösung im Mittelpunkt, ferner sollen auch nicht-funktionale Kriterien, wie die Lösungsdauer und Anzahl an durchgeführten Operationen, in den Vergleich einfließen.
Diese Kriterien werden auch dazu herangezogen, um die prinzipielle Eignung des in dieser Arbeit entwickelten kommunikations- und ereignisbasierten Ansatzes zur stabilen Netzregelung zu evaluieren. So soll gezeigt werden, dass auch ein kommunikations-basierter Ansatz in der Lage ist, das Netz stabil zu steuern.

10.3.2. Vorgehen

Aufbauend auf den in Abschnitt 10.2 beschriebenen Szenarien wurden mithilfe des Opta Planner Benchmark Tool mehrere Testläufe durchgeführt. Innerhalb eines Testlaufs wurden alle Szenarien mit jeweils beiden Algorithmen betrachtet.
Hierbei ermittelt Opta Planner für jedes Szenario, welchen Score der gerade betrachtete Algorithmus erzielen konnte. Auch weitere Messwerte, wie z.B. die Lösungsdauer, werden durch das Benchmark Tool ermittelt und anschließend als HTML-Reports gespeichert.

10.3.3. Ergebnisse

Vergleicht man die beiden betrachteten Algorithmen bezüglich ihres, in den Testläufen und je Szenario, erreichten Scores so fällt die Betrachtung zu Gunsten von *Best Fit* aus.

Solver	szenario1	szenario2	szenario3	szenario4	szenario5	emptyBalancingProblem	Average	Ranking
FirstFit only 1	0hard/0soft 0	0hard/-50000soft 1	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/-8334soft	1
BestFit only 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft 0	0hard/0soft	0

Abbildung 10.6.: Übersicht über die in einem Testlauf erzielten Scores pro Algorithmus und Szenario.

In Abbildung 10.6 sind die für jedes Szenario erreichten Score-Werte pro Algorithmus für einen Testlauf dargestellt. Entgegen der initialen Erwartungen konnte *Best Fit* in Szenario 2 die optimale Lösung finden. Der Algorithmus konnte dabei einen Score von 0hard/0soft erzielen und lag damit vor *First Fit*, welcher einen Score von 0hard/-50000soft erzielte.
Dieses Ergebnisses, legt die Schlussfolgerung nahe, in jedem Fall *Best Fit* als Construction Heuristic zu verwenden. In diesem Fall hätte man nur das Kriterium der Lösungsgüte zur Bewertung herangezogen. Da die Lösung aber auch in gewissen zeitlichen Schranken erfolgen muss, um nicht wertlos zu sein, und ggf. die Lösung noch mit einer Local Search verbessert werden kann, ist es nicht

ausreichend den Score als alleiniges Bewertungskriterium anzulegen.

Legt man zusätzlich die Dauer zur Lösung des Problems und, als Maß für die Skalierbarkeit, die Anzahl an Operationen pro Sekunde an, so ergibt sich ein anderes Bild.

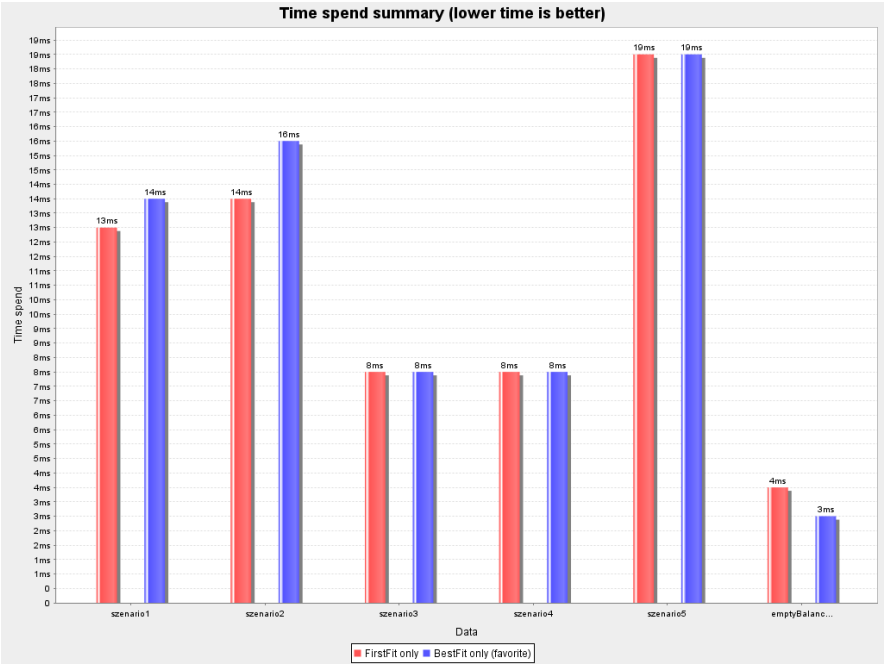

Abbildung 10.7.: Übersicht über die Dauer, die die Algorithmen zur Lösung der Szenarien benötigen.

Sowohl bei der Lösungsdauer (vgl. Abbildung 10.7), als auch bei den Rechenoperationen (vgl. Abbildung 10.8) liegt *First Fit*, wenn auch knapp, vor *Best Fit*. Je nach Testlauf kann der Unterschied auch größer ausfallen (vgl. Abbildung 10.7 und Abbildung 10.9).

10.3.4. Schlussfolgerungen

Ausgehend von den im vorherigen Abschnitt beschriebenen Ergebnissen ist es nicht möglich, eine allgemeingültige Empfehlung für eine Construction Heuristic zur Netzsteuerung zu geben. Auf Grund des besseren Scores, auch bei schwierigen Szenarien, sollte man für weitere Untersuchungen zuerst mit einem zur Nutzung von *Best Fit* konfigurierten Solver beginnen. Sobald man jedoch in dem Bereich des Real-Time Planning mit Local Search geht, lautet die Empfehlung auf *First Fit* zu setzen, um schnell eine initiale Lösung zu erhalten.

Ein anderer Aspekt, der auch bei der Wahl der Construction Heuristic betrachtet werden sollte, wurde während der Durchführung von Integrationstests zwischen Simulationsumgebung und CGMU entdeckt. In dieser Konstellation werden ständig neue Ereignisse an die CGMU geschickt und so die Planning Facts verändert. Die CGMU führt also ein Real-Time Planning durch.

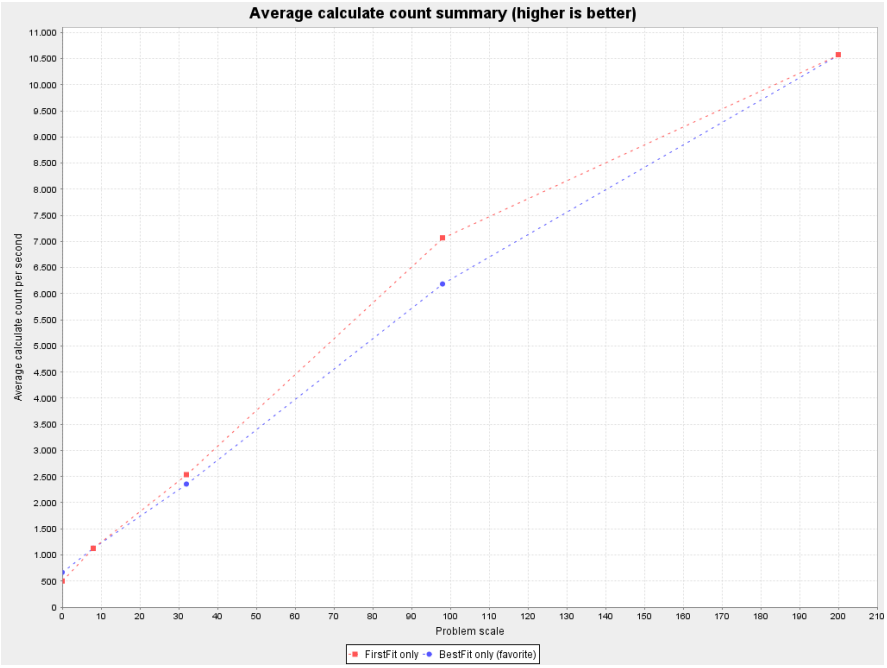

Abbildung 10.8.: Übersicht über die Anzahl an Rechenschritte pro Sekunde und Szenario.

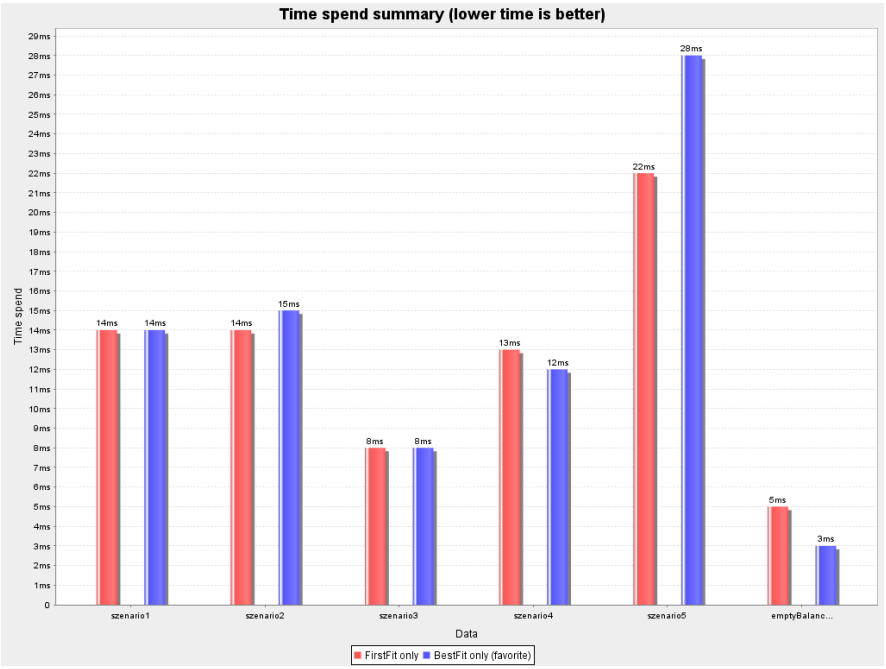

Abbildung 10.9.: Übersicht eines Testlaufs mit einer größeren zeitlichen Abweichung zwischen den Algorithmen.

Sobald Opta Planner die durch die Ereignisse induzierten Änderungen in die Solution übernommen hat, wird der Plannungsprozess erneut gestartet. Zunächst läuft also die Construction Heuristic und anschließend ggf. die Local Search Phase. Hierbei konnte das Verhalten beobachtet werden, dass bereits die Construction Heuristic eine sinnvolle Lösung fand und diese dann dazu verwendet wurde, das Netz zu stabilisieren. Die Local Search Phase war daher überflüssig bzw. konnte sogar zum Absturz führen, wenn sie durch eine ungeeignete Konfiguration in dem Lösungsprozess stecken blieb und keinen Fortschritt mehr erzielen konnte. Daher wurden die praktischen Integrationstests in diesem Fall nur mit einem *First Fit* oder *Best Fit* Solver durchgeführt.
Dies zeigt, dass pauschale Aussagen im Bereich der Planungsalgorithmen nicht getroffen werden können. Vielmehr muss die Konfiguration an die Szenarien angepasst werden. Für die hier prototypisch betrachteten Szenarien ist die Verwendung einer Construction Heuristic der optimale Weg, um eine automatisierte Netzsteuerung zu erreichen. Für praxisnahe Szenarien oder einen realen Einsatz muss dies nicht zwangsläufig zutreffen. Um das Ziel eines Einsatzes in der Praxis erreichen zu können, müssen die Szenarien immer weiter vergrößert, die Konfiguration angepasst und die Ergebnisse überprüft werden. Während dieses langwierigen Prozesses treten oft Situationen und Probleme auf, die im Vorfeld so nicht vorhergesehen werden konnten.

10.3.5. Abschätzung der Praxistauglichkeit

Um eine Abschätzung treffen zu können, wie praxistauglich der entwickelte Steuerungsmechanismus ist, soll an dieser Stelle die Zeitverzögerung zur Ermittlung und Durchführung der Steuerungsmaßnehmen berechnet werden. Dazu werden die folgenden Messgrößen benötigt:
Bei den in Tabelle 10.1 dargestellten Werten handelt es sich um Maximalwerte. Zum Beispiel wird für

Messgröße	Symbol	Wert
Round Trip Tip	RTT	<50ms
Planungsdauer	PD	<220ms
Verarbeitung CGMU	V_{CGMU}	100ms
Verarbeitung Netzteilnehmer	V_{Netz}	100ms

Tabelle 10.1.: Übersicht über die zur Abschätzung benötigte Messgrößen

die Dauer der Durchführung des Planungsprozesses ein Wert von 220ms angenommen. Dieser wurde bei der Betrachtung von Szenario 5, dem mit Abstand aufwändigsten Szenario, mit eingeschalteter Debug-Option ermittelt. Wie den Werten aus den Abbildungen 10.7 und 10.9 entnommen werden kann, handelt es sich bei dem für die Abschätzung angenommenen Wert um ein Worst Case Szenario. Der angenomme Wert ist ca. um den Faktor 8 größer, als die bei den Benchmarks ermittelten Werte. Das Gleiche gilt auch für die Round Trip Time, welche innerhalb von Deutschland typischerweise bei unter 50ms[1] liegt. Als Verzögerung für die Verarbeitung der Ereignisse wurden pauschal 100ms

[1] `https://de.wikipedia.org/wiki/Paketumlaufzeit`

angenommen und sie somit sehr großzügig ausgelegt.
Zur Berechnung der Gesamtverzögerung werden die einzelnen Messgrößen folgendermaßen addiert:

$$Gesamtverzögerung = RTT + PD + V_{CGMU} + V_{Netz} \tag{10.1}$$

$$Gesamtverzögerung = 50ms + 220ms + 100ms + 100ms \tag{10.2}$$

$$Gesamtverzögerung = 470ms \tag{10.3}$$

Ergebnis der Abschätzung

Die Betrachtung des Worst Case Szeanrios ergab eine Gesamtverzögerung von 470ms, also knapp einer halben Sekunde.
Bei der Steuerung des Stromnetzes durch die Netzbetreiber wird auf 3 Regelstufen[2][3] zurückgegriffen:

- Primärregelung
- Sekundärregelung
- Minutenreserve

Die Primärregelung besitzt die gleiche Aufgabe wie die CGMU, nämlich das Netz wieder zu stabilisieren. Die anderen Regelstufen werden für den Ausgleich größerer Schwankungen benötigt. Alle Regelstufen unterscheiden sich dabei in ihrer Aktivierungsdauer.
Bei der durch die CGMU umgesetzten Steuerung handelt es sich um eine Primärregelung. Die Primärregelung muss dabei innerhalb von 30 Sekunden zur Verfügung stehen[4]. Somit ist die durch die CGMU implementierte Steuerung, mit ihrer Reaktionszeit von unter einer Sekunde, dafür geeignet.

Hiermit kann auch die durch Croft u. a. (2011) und Zeilinger (2011) aufgestellte These, dass kommunikationsbasierte Ansätze nicht schnell genug sind, widerlegt werden.
Selbst bei der Betrachtung von Worst Case Szenarien und dem Einsatz von Standardhardware (ein Laptop mit Intel Core i7 2,7GHz und 8GB RAM) lag die Verzögerung deutlich unter den geforderten 30 Sekunden.
Der in dieser Arbeit präsentierte Ansatz hat den Vorteil, dass durch das zentrale Wissen der CGMU eine bessere Koordination der einzelnen Netzteilnehmer erreicht werden kann. Daraus sollte eine bessere Ausnutzung von Ressourcen und eine qualitativ hochwertigere Steuerung folgen, als sie Ansätze mit *Dynamic Demand Control* (vgl. Abschnitt 3.2) erreichen können. Diese Hypothese ist aber durch zukünftige Untersuchungen erst noch zu belegen.

[2] `http://www.amprion.net/primaerregelung-sekundaerregelung-minutenreserve`
[3] `http://www.apg.at/de/markt/netzregelung`
[4] `http://www.amprion.net/primaerregelung-sekundaerregelung-minutenreserve`

10.4. Zusammenfassung

Mit den in diesem Abschnitt durchgeführten Untersuchungen konnte ein Vergleich zwischen den beiden Construction Heuristiken *Best Fit* und *First Fit* gezogen werden. Dabei wurde festgestellt, dass *First Fit* mehr Operationen pro Sekunde durchführt, dafür *Best Fit* optimale Lösungen für Szenarien findet, bei denen *First Fit* dies nicht schafft. Aus diesem Grund sollte *First Fit* in Kombination mit einer *Local Search* dazu genutzt werden um Real-Time Planning zu realisieren. *Best Fit* eignet sich auch für den alleinigen Einsatz, weshalb die Netzsteuerung zunächst auf diesem Algorithmus basiert.

Danach wurde aufbauend auf den Szenarien sowie den Benchmark-Ergebnissen eine Abschätzung der Planungsverzögerung vorgenommen. Über die Abschätzung soll ermittelt werden, ob das System schnell genug reagiert. Da die Gesamtverzögerung unter einer Sekunde lag und die Primärregelung in der manuellen Netzsteuerung eine Reaktionszeit von bis zu 30 Sekunden vorsieht, erfüllt die hier entwickelte Steuerung diese zeitlichen Anforderungen.
Hiermit konnte auch gezeigt werden, dass kommunikations-basierte Ansätze sich sogar gut für den Einsatz zur Netzsteuerung eignen, da die Verzögerung deutlich unter den geforderten 30 Sekunden liegt. Neben *Dynamic Demand Control* können also auch kommunikations-basierte Ansätze zur Netzsteuerung verwendet werden. Und daraus folgt, dass sich auch der Einsatz einer ereignisgesteuerten Architektur bewährt hat.

11. Nachbetrachtung

In Kapitel 5 wurden die, an das zu entwickelnde System gestellten, funktionalen und nichtfunktionalen Anforderungen formuliert (vgl. die Abschnitte 5.2 bis 5.4). Nach der Realisierung des Systems soll in diesem Kapitel der Erfüllungsgrad der einzelnen Anforderungen betrachtet und so eine zusammenfassende Bewertung des entwickelten Systems erfolgen.

Hierzu wird der Erfüllungsgrad der einzelnen Anforderungen tabellarisch veranschaulicht. Außerdem werden die Hintergründe für nicht vollständig erfüllte Anforderungen erläutert und weitere Schritte zur Erfüllung der Anforderung vorgestellt.

11.1. Übersicht

In den Tabellen 11.1 und 11.2 werden alle an das System gestellten Anforderungen aufgelistet.
Diese sind mit ihrem Kürzel und dem jeweiligen Erfüllungsgrad dargestellt. Zusätzlich zeigen die Tabellen, ob ggf. Realisierungsaspekte oder architektonische Aspekte für die unvollständige Umsetzung der Anforderung verantwortlich sind.
In der Spalte *Erfüllungsgrad* symbolisiert ein Häkchen eine vollständig erfüllte Anforderung. Ein Kreuz steht für eine nicht oder nicht vollständig erfüllte Anforderung.

11.1.1. Funktionale Anforderungen

Wie Tabelle 11.1 zeigt, konnten, bezogen auf die Simulationsumgebung und das Monitoring, alle funktionalen Anforderungen erfüllt werden.
Bei den an die *Central Grid Management Unit* gestellten Anforderungen wurde die Steuerung der Netzfrequenz nicht vollständig umgesetzt. Dies lag zum einen an der Komplexität der Realisierung und zum anderen an der Veränderung des Ansatzes zur Steuerung des Stromnetzes. Aus architektonischer Sicht sind jedoch alle Voraussetzungen zur Erfüllung der Anforderungen gegeben.

Im Verlauf der Realisierung wurde zur Vereinfachung die Differenz zwischen produzierter und verbrauchter Energie als Indikator zum Starten eines Planungsprozesses genutzt. Die Netzfrequenz wurde ausgeblendet.
Auf Grund des überraschend schnellen Planungsprozess, konnte die Idee einer iterativen Steuerung des Stromnetzes verfolgt werden. Hierbei wird das Stromnetz so gesteuert, dass die Differenz

zwischen Energieproduktion und -verbrauch zu jedem Zeitpunkt Null ist. Hierdurch konnte erreicht werden, dass sich die Netzfrequenz in der Simulation nicht ändert und daher vernachlässigt werden kann. Eine Betrachtung dieses Aspekts in einem realen Einsatz ist obligatorisch. Dieses gilt auch für andere Qualitätsmerkmaler der Stromversorgung (z.B. der Blindleistung), welche in weiterführenden Untersuchungen genauer zu betrachten sind.
Eine Frequenz gestützte Steuerung ist darüber hinweg deutlich aufwändiger zu realisieren, da diese inkrementell über die Zeit und in Abhängigkeit von der Ausgangsfrequenz berechnet werden muss. Daher wurde in dieser Arbeit bei der prototypischen Umsetzung zunächst darauf verzichtet.

Anforderung	Erfüllungsgrad	Realisierung	Architektur
SF-1)	✓	-	-
SF-2)	✓	-	-
SF-3)	✓	-	-
SF-4)	✓	-	-
SF-5)	✓	-	-
SF-6)	✓	-	-
SF-7)	✓	-	-
SF-8)	✓	-	-
SF-9)	✓	-	-
SF-10)	✓	-	-
SF-11)	✓	-	-
MF-1)	✓	-	-
MF-2)	✓	-	-
MF-3)	✓	-	-
MF-4)	✓	-	-
MF-5)	✓	-	-
CF-1)	✓	-	-
CF-2)	✓	-	-
CF-3)	✓	-	-
CF-4)	✓	-	-
CF-5)	×	✓	-
CF-6)	✓	-	-
CF-7)	✓	-	-
CF-8)	×	✓	-
CF-9)	✓	-	-

Tabelle 11.1.: Übersicht funktionale Anforderung und ihr Erfüllungsgrad

Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen standen bei der Umsetzung nicht im Zentrum der Arbeit. Viel mehr lag der Schwerpunkt darin einen funktionierenden Prototypen des Systems zu entwickeln und so die zentralen Probleme und Fragestellungen des Anwendungsbereichs zu lösen. Trotz dieser

Ausrichtung konnten durch den Entwurf und die Realisierung der Großteil an nichtfunktionalen Anforderungen erfüllt werden.

Das entwickelte System weist jedoch noch im Bereich der Zuverlässigkeit Verbesserungsbedarf auf. Da es sich bei dem Stromnetz um eine kritische Infrastruktur handelt, sind besonders bezogen auf die Zuverlässigkeit weiterführende Untersuchungen und Verbesserungen vorzunehmen.
Um den Ausfall von Netzteilnehmern zeitnah erkennen zu können, könnten zusätzliche *Heartbeat*-Ereignisse eingeführt werden. Diese werden von Netzteilnehmern periodisch an die CGMU gesendet. Bleibt ein *Heartbeat*-Ereignis eines Netzteilnehmers über eine längere Zeit aus, erkennt die CGMU dass der Netzteilnehmer ausgefallen ist und kann darauf reagieren. Mit Hilfe der Fähigkeit von *Jboss Drools* zeitliche Muster in Ereignisfolgen erkennen zu können, lässt sich ein *Heartbeat* effizient umsetzen.
Für die Realisierung wurde auf eine redundante Auslegung der Nachrichteninfrastruktur und der CGMU verzichtet. Da es sich bei beiden Komponenten um neuralgische Knotenpunkte des Systems handelt, müssen sie vor Ausfällen abgesichert werden. Hierzu könnte man mehrere Instanzen parallel betreiben, die sich ggf. auch gegenseitig kontrollieren. Oder man hält eine weitere Instanz als Backup vor, die bei einem Ausfall des primären Systems dessen Aufgabe übernimmt. Auch für diesen Aspekt gilt es, in weiterführenden Betrachtungen, Konzepte zu evaluieren und geeignete Lösungen zu entwickeln.

Anforderung	Erfüllungsgrad	Realisierung	Architektur
NF1)	✓	-	-
NF2)	×	✓	-
NF3)	×	✓	-
NF4)	✓	-	-
NF5)	✓	-	-
NF6)	✓	-	-
NF7)	✓	-	-
NF8)	✓	-	-
NF9)	✓	-	-
NF10)	✓	-	-
NF11)	✓	-	-
NF12)	✓	-	-

Tabelle 11.2.: Übersicht nichtfunktionale Anforderung und ihr Erfüllungsgrad

11.2. Zusammenfassung

Im Nachgang kann festgestellt werden, dass obwohl der Fokus auf einer prototypischen Implementierung des Systems lag, der Großteil der funktionalen und nichtfunktionalen Anforderungen erfüllt werden konnte.

Vor einem Praxiseinsatz sind noch weitere Untersuchungen im Bereich der Frequenzsteuerung und der Ausfallsicherheit nötig. Hierbei ist eine stabile Steuerung der Netzfrequenz zu entwickeln und es müssen Maßnahmen ermittelt werden, um einen Ausfall der Nachrichteninfrastruktur und der CGMU kompensieren zu können.
Da die Gründe zur Nichterfüllung der Anforderungen im Bereich der Realisierung liegen und die Architektur die Umsetzung aller Anforderungen unterstützt, kann insgesamt ein sehr positives Fazit bezüglich der Anforderungserfüllung des Systems gezogen werden.

12. Zusammenfassung und Ausblick

Zum Abschluss dieser Arbeit sollen das in der Arbeit verwendete Vorgehen und die erzielten Ergebnisse zusammenfassend betrachtet werden. Darauf folgt ein Ausblick auf Möglichkeiten zur Fortführung der betrachteten Themen. Dabei werden sowohl Ansätze zur Erweiterung der Simulationsumgebung, als auch zur Verbesserung der Steuerungs- und Planungsprozesse erläutert.

12.1. Zusammenfassung

Das Ziel dieser Arbeit bestand in der Entwicklung einer ereignisgesteuerten Architektur für Demand Side Management in Smart Grids.

Die Ziele und die mit ihnen verbundenen Hypothesen werden in Kapitel 1 beschrieben. Im Verlauf der Arbeit konnten diese vollständig erfüllt und bestätigt werden.
Kapitel 2 widmete sich der technischen Sicht auf die Steuerung und Abläufe innerhalb von Stromnetzen. Die physikalischen Grundlagen wurden erläutert. Demand Side Management wurde konkret definiert.
Im Anschluss (Kapitel 3) wurden Ansätze aus anderen Arbeiten vorgestellt. Es wurde dabei herausgearbeitet, was die zentralen Merkmale der Ansätze sind und welche Fragestellung sie zu beantworten versuchen. Außerdem wurde ein Vergleich zwischen dem in den verwandten Arbeiten genutzten und den für die Arbeit vorgesehenen Ansatz gezogen, um die Ansätze voneinander abgrenzen zu können. Im Rahmen der Literaturrecherche konnte keine Arbeit gefunden werden, die ebenfalls eine ereignisgesteuerte Architektur zur Simulation bzw. Steuerung von Stromnetzen nutzt.

Die Kapitel 4 bis 7 befassten sich mit dem eigentlichen Entwurf des Netzes und der Teilsysteme. Kapitel 4 skizzierte den Aufbau des Netzes aus informationstechnischer Sicht. Die verschiedenen Teilsysteme mit ihren unterschiedlichen Aufgaben wurden eingeführt und das Zusammenspiel der Systeme über die Kommunikationsinfrastruktur vorgestellt.
Im folgenden Kapitel 5 wurden die funktionalen und nicht-funktionalen Anforderungen anhand der Aufgabenbeschreibungen abgeleitet.
Aufbauend auf den ermittelten Anforderung erfolgte in den Kapiteln 6 und 7 der Komponentenentwurf der Teilsysteme. Die Steuerung mittels Ereignissen stellt dabei das zentrale Konzept über alle Entwurfsebenen dar. Als Mittel zur Verbindung sowohl von Komponenten als auch von Teilsystemen, ermöglicht sie es ein System aus extrem lose gekoppelten Komponenten zu erstellen, das sich einfach

erweitern und anpassen lässt.

Die Entwürfe wurden anschließend auf den verschiedenen Plattformen umgesetzt. Mit JSON als Nachrichtenformat konnte eine Brücke zwischen der .NET- und der JAVA-Plattform geschlagen werden, so dass die Simulationsumgebung und die Central Grid Management Unit über das fachliche Ereignismodell miteinander kommunizieren können.
Um Steuerungsmaßnahmen automatisch ermitteln zu können, wurde das Planungsproblem, Verbrauchs- und Produktionsmengen angleichen zu müssen, mit Hilfe von Jboss Opta Planner als Fahrplan von Steuerungsmaßnahmen modelliert. Anschließend wurden die dabei zu berücksichtigenden Regeln als Constraints in Jboss Drools formuliert, so dass ein iterativer und damit schnellerer Planungsprozess erreicht werden konnte.

Schon bei der Implementierung der Teilsysteme wurde testgetrieben vorgegangen um durch die lose Kopplung entstehende Fehler frühzeitig erkennen und beheben zu können. Kapitel 9 beschreibt die weiteren Maßnahmen für den Test der Software und die Verifikation der Funktionalität der Teilsysteme. So ist neben der Zustands- die Verhaltensverifikation für ereignisgesteuerte Systeme sehr hilfreich, da sich mit ihr asynchrones Verhalten einfach überprüfen lässt. Die Tests müssen so nicht durch Warteanweisungen verzögert werden.

Zum Abschluss der Arbeit wurden in Kapitel 10 verschiedene Planungsalgorithmen anhand vorher definierter Szenarien verglichen. Hierbei wurde festgestellt, dass sich *First Fit* in Verbindung mit einer Local Search besonders für den Einsatz in Real-Time Planning eignet. Dahingegen kann *Best Fit* zum Erreichen einer höheren Lösungsgenauigkeit genutzt werden, da es auch in schwierigen Szenarien die optimale Lösung in angemessener Zeit findet. So läuft aktuell nach dem aktuellen Entwurf auch die Netzsteuerung auf Basis des *Best Fit*-Algorithmus.
Die zur Bewertung genutzten Benchmark-Ergebnisse wurden zusätzlich für eine Abschätzung über die Praxistauglichkeit der entwickelten Netzsteuerung genutzt. Da eine Verzögerung von unter einer halben Sekunde in einer Worst Case Abschätzung ermittelt wurde, kann die Steuerung die Anforderung von einer Reaktionszeit von 30 Sekunden erfüllen. Diese ist als maximale Verzögerung für eine Primärregelung vorgesehen. Ferner konnte damit auch widerlegt werden, dass kommunikationsbasierte Ansätze für eine automatische Netzsteuerung zu langsam sind.
Kapitel 11 betrachtet anschließend welche Anforderungen durch das System abgedeckt werden konnten. Für die nicht erfüllten Anforderungen werden die Gründe, die dazu führten, genannt. Auf diese Weise wird das erstellte System zum Abschluss bewertet.

12.2. Ausblick

Der erste Schritt, um an dieser Arbeit anzuknüpfen, liegt in der Überführung des Systemprototypen hin zum produktiven Einsatz. Das dazu nötige Vorgehen wurde bereits in Abschnitt 10.3.4 aufgezeigt. So müssen die Szenarien kontinuierlich erweitert werden. Anschließend sind die Konfigurationen und Regeln darauf zu prüfen, ob die Anforderungen an den Planungsprozess noch immer erfüllt werden. Im Rahmen dieser Arbeiten sollte auch die Simulationsumgebung um neue Netzteilnehmer und Modelle erweitert werden.

12.2.1. Simulationsumgebung

Ein weiterer, die Simulationsumgebung betreffender Punkt, stellt die Portierung der Umgebung und deren Anpassung an den Einsatz auf einem Microsoft PixelSense dar. Mit der Implementierung auf Basis von .NET und C#, sowie der Konzentration auf Drag&Drop als Bedienkonzept, wurden hierfür die zentralen Voraussetzungen bereits gelegt. Dennoch werden im Verlauf der Portierung wahrscheinlich weitere Anpassungen nötig sein.

Weiterhin ist bei der Weiterentwicklung der Simulationsumgebung der Aspekt der Wissensvermittlung zu berücksichtigen und weiter auszugestalten. Dazu ist es nötig, ein Konzept für die Wissensvermittlung zu entwickeln und dieses im Anschluss auf dessen Eignung empirisch zu untersuchen. So kann sichergestellt werden, dass die Simulationsumgebung ihre Aufgaben in diesem Bereich auch im vorgesehenen Umfang erfüllt.

12.2.2. Stromnetzsteuerung

Weitere Ansatzpunkte für Verbesserungen liegt in der Steuerung des Stromnetzes. Zunächst wäre eine genauere Betrachtung des Frequenzverlaufs und die Einbeziehung der Frequenz in den Steuerungsprozess notwendig. Dies ist mit der Hoffnung verbunden, den Steuerungsprozess qualitativ zu verbessern. Im selben Schritt kann auch ein Vergleich mit den Ergebnissen der Arbeiten aus dem Bereich des *Dynamic Demand Control* (Croft u. a. (2011) und Zeilinger (2011)) gezogen werden. Hierbei wäre zu untersuchen, ob eine zentrale Steuerung, gegenüber der verteilten autonomen Reaktion von Netzteilnehmern, Vorteile bei der Ressourcenausnutzung und der Qualität der Netzsteuerung bietet.

12.2.3. Predictive Analytics und Data Mining

Eine andere Möglichkeit die Steuerung weiterzuentwickeln und ggf. noch besser zu machen, liegt in der Weiterentwicklung des reaktiven Systems hin zu einem proaktiv arbeitenden. Im ersten Schritt

könnten bereits bestehende Lastkurven und -pläne in die Steuerung einbezogen werden, welche auch für die bisherige manuelle Steuerung genutzt werden. Dadurch, dass nicht mehr jede Veränderung betrachtet werden muss, ergeben sich möglicherweise schon Verbesserungen.
In einem weiteren Schritt kann die zugrundeliegende ereignisgesteuerte Architektur noch weiter einbezogen werden. Dieses könnte durch die Analyse der bestehenden Lastpläne im Hinblick auf wiederkehrende Muster oder Situationen erfolgen. Nach deren Identifikation könnten sie in Regeln zur Ereignisverarbeitung umgesetzt werden, so dass noch gezielter reagiert werden kann.
Außerdem kann auch die Menge an beobachteten Ereignissen ausgenutzt werden. Hierzu werden die Ereignisse im Event Store gespeichert und mit Methoden aus den Bereichen Predictive Analytics und Data Mining analysiert. Die dabei gewonnen Erkenntnisse fließen anschließend in den Steuerungsprozess mit ein.
Interessant ist auch der Bereich des *Proactive Event Processing*. In diesem Bereich wird versucht das Auftreten von Ereignissen vorherzusagen und dann proaktiv zu reagieren. Hierbei kommen u.a. Methoden aus dem Bereich der Predictive Analytics und der Statistik zum Einsatz, da das zukünftige Auftreten von Ereignissen mit Unsicherheit behaftet ist. Dieser Bereich der Ereignisverarbeitung unterliegt im Moment einem großen Forschungsinteresse[1].
Ein auf diese Weise erweitertes System wäre dann nicht nur in der Lage direkt auf Ungleichgewichte innerhalb des Stromnetzes zu reagieren, sondern könnte Anzeichen dafür im Voraus erkennen und daher vorausschauende Maßnahmen ergreifen.

[1] `http://www.orgs.ttu.edu/debs2013/presentations/proactiveEPinAction.pdf`

Anhang

A. Fachliches Datenmodell

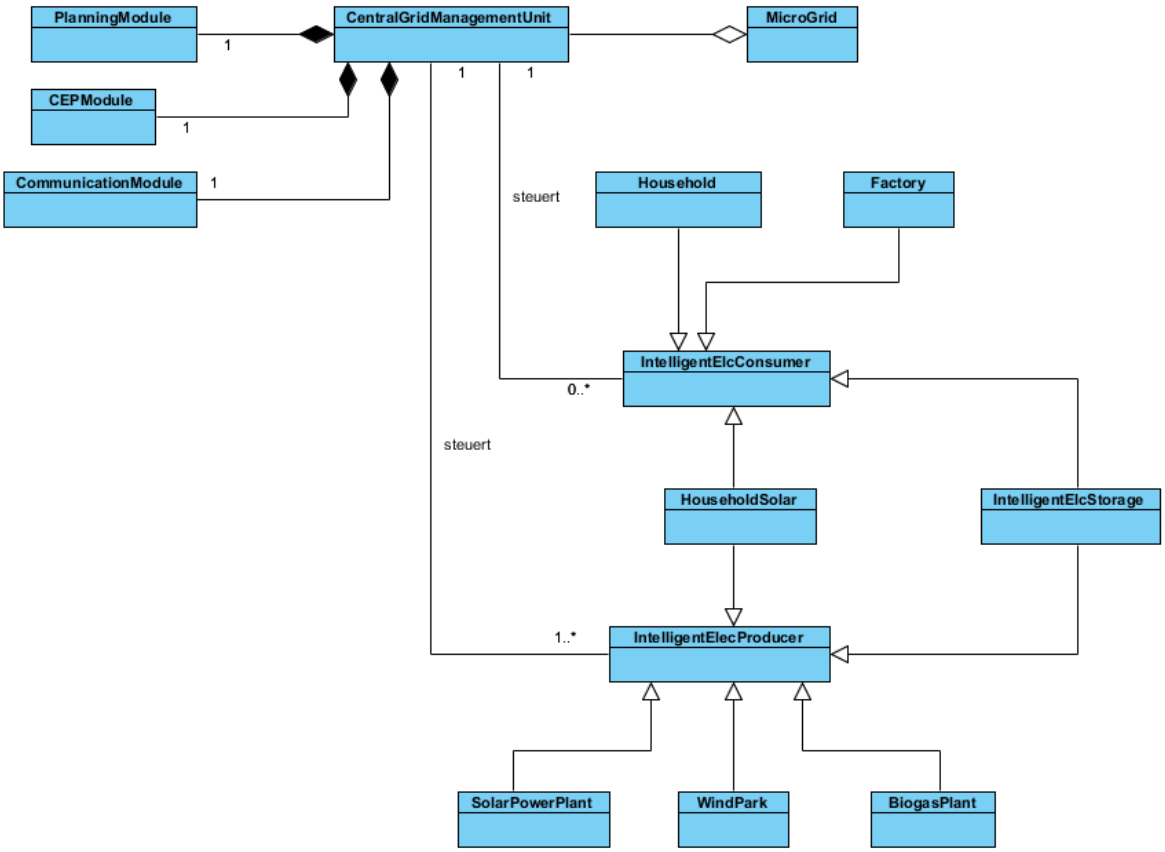

Abbildung A.1.: Fachliches Datenmodell des Micro-Grids

B. Lastprofil

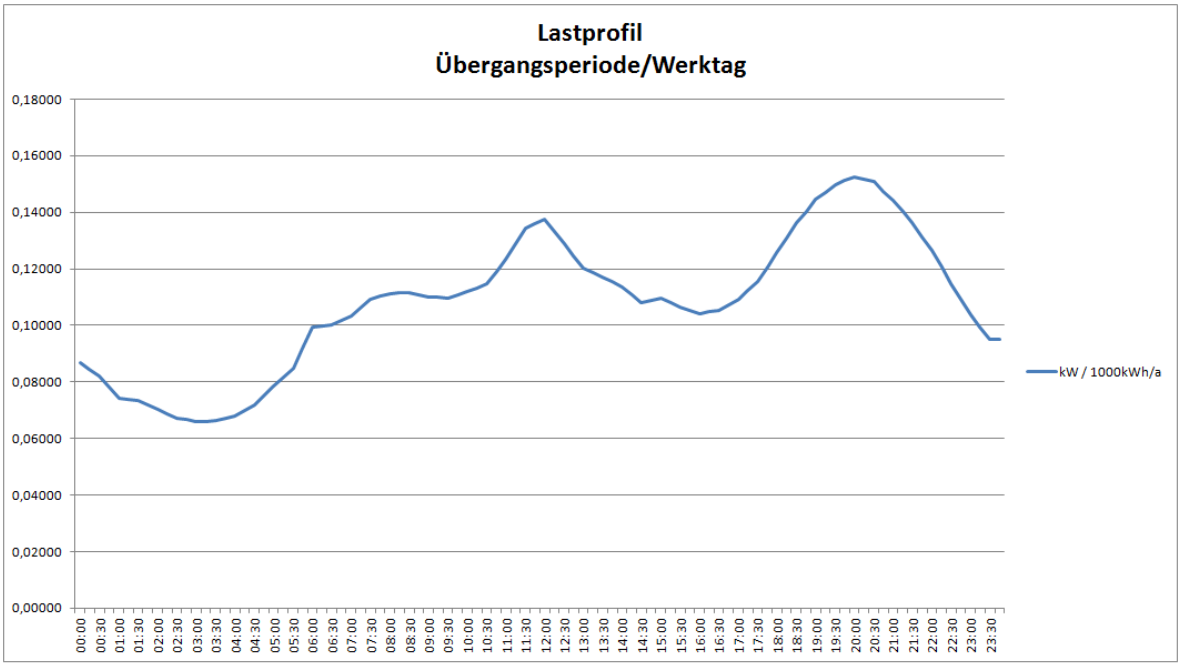

Abbildung B.1.: Lastprofil eines Werktags in der Übergangsperiode

C. Szenarien zur Evaluierung von Planungsalgorithmen

```
public abstract class ProductionConsumptionBalancingGenerator {

   public static ProductionConsumptionBalancing CreateBalancingProblem(long
       totalProduction, long totalConsumption, List<BaseOfferEvent> offerEvents)
   {

      ProductionConsumptionBalancing problem = new ProductionConsumptionBalancing();

      // Initialize problem facts
      problem.setTotalElectricityProduction(new
          TotalEnergyProduction(totalProduction));
      problem.setTotalElectricityConsumption(new
          TotalEnergyConsumption(totalConsumption));
      problem.setOffers(offerEvents);

      /*
       * Create as much schedule items and dummies as offer events
       */

      int countOffers = offerEvents.size();

      List<ScheduleItem> schedule = new ArrayList<ScheduleItem>();
      List<PlaceholderOfferEvent> dummies = new ArrayList<PlaceholderOfferEvent>();

      ScheduleItem item = null;
      PlaceholderOfferEvent dummy = null;

      // Initialize items
      for (int i = 0; i < countOffers; i++) {
         item = new ScheduleItem(i);
         dummy = new PlaceholderOfferEvent(i);
```

```
        schedule.add(item);
        dummies.add(dummy);

    }

    // Set initialize list as schedule in solution
    problem.setSchedule(schedule);

    // Set dummy list
    problem.setDummies(dummies);

    return problem;

}

/**
 * Scenario 1:
 * Production: 1000kW
 * Consumption: 750kW
 * Prodction Offers: 50kW, -200kW
 * Consumption Offers: 50kW, 150kW
 * Solveable: yes
 * @return The constructed problem.
 */
public static ProductionConsumptionBalancing createScenario1() {

    ProductionConsumptionBalancing problem = null;

    final String entity = "id";

    // Problem facts
    long totalProduction = 1000000; // [1000 kW]
    long totalConsumption = 750000; // [750 kW]

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();

    // Consumption offers: [50 kW], [150 kW]
    events.add(new ConsumptionOfferEvent(0, entity, 50000, new
        NonThrottledEvent(0, entity)));
    events.add(new ConsumptionOfferEvent(1, entity, 150000, new
        NonThrottledEvent(1, entity)));
```

```
    // Production offers: [50 kW], [-200 kW]
    events.add(new ProductionOfferEvent(2, entity, 50000, new
        NonThrottledEvent(2, entity)));
    events.add(new ProductionOfferEvent(3, entity, -200000, new
        NonThrottledEvent(3, entity)));

    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);

    return problem;

  }

  /**
   * Scenario 2:
   * Production: 1000kW
   * Consumption: 750kW
   * Prodction Offers: 500kW, -300kW, 200kW
   * Consumption Offers: -700kW, 600kW, 300kW, -50kW
   * Solveable: no
   * Best Solution: [CP 300kW]
   * @return The constructed problem.
   * Problem: Overlocked that only one production event and one consumption event
       per time
   */
  public static ProductionConsumptionBalancing createScenario2() {

    ProductionConsumptionBalancing problem = null;

    final String entity = "id";

    // Problem facts
    long totalProduction = 1000000; // [1000 kW]
    long totalConsumption = 750000; // [750 kW]

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();

    // Consumption offers: [-700 kW], [600 kW], [300 kW], [-50kW]
    events.add(new ConsumptionOfferEvent(0, entity, -700000, new
        NonThrottledEvent(0, entity)));
    events.add(new ConsumptionOfferEvent(1, entity, 600000, new
        NonThrottledEvent(1, entity)));
```

```
    events.add(new ConsumptionOfferEvent(2, entity, 300000, new
        NonThrottledEvent(2, entity)));
    events.add(new ConsumptionOfferEvent(3, entity, -50000, new
        NonThrottledEvent(3, entity)));


    // Production offers: [500 kW], [-300 kW], [200 kW]
    events.add(new ProductionOfferEvent(4, entity, 500000, new
        NonThrottledEvent(4, entity)));
    events.add(new ProductionOfferEvent(5, entity, -300000, new
        NonThrottledEvent(5, entity)));
    events.add(new ProductionOfferEvent(6, entity, 200000, new
        NonThrottledEvent(6, entity)));

    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);

    return problem;

  }

  /**
   * Scenario 3:
   * Production: 1000kW
   * Consumption: 0kW
   * Prodction Offers: -500kW, -1000kW
   * Consumption Offers:
   * Solveable: yes
   * Solution: [CP -1000kW]
   * @return The constructed problem.
   */
  public static ProductionConsumptionBalancing createScenario3() {

    ProductionConsumptionBalancing problem = null;

    final String entity = "id";

    // Problem facts
    long totalProduction = 1000000; // [1000 kW]
    long totalConsumption = 0; // [0 kW]

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();
```

```
    // Production offers: [-500 kW], [-1000 kW]
    events.add(new ProductionOfferEvent(1, "Prod1", -500000, new
        NonThrottledEvent(0, entity)));
    events.add(new ProductionOfferEvent(2, "Prod1", -1000000, new
        NonThrottledEvent(1, entity)));

    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);

    return problem;

}

/**
 * Scenario 4:
 * Production: 0kW
 * Consumption: 750kW
 * Prodction Offers:
 * Consumption Offers: -350kW, -750kW
 * Solveable: yes
 * Solution: [C0 -750kW]
 * @return The constructed problem.
 */
public static ProductionConsumptionBalancing createScenario4() {

    ProductionConsumptionBalancing problem = null;

    final String entity = "id";

    // Problem facts
    long totalProduction = 0; // [1000 kW]
    long totalConsumption = 750000; // [0 kW]

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();

    // Production offers: [-350 kW], [-750 kW]
    events.add(new ConsumptionOfferEvent(0, "Con1", -350000, new
        NonThrottledEvent(0, entity)));
    events.add(new ConsumptionOfferEvent(1, "Con1", -750000, new
        NonThrottledEvent(1, entity)));
```

```
    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);

    return problem;

  }

  /**
   * Scenario 5:
   * Production: 1000kW
   * Consumption: 750kW
   * Prodction Offers: -150kW, 50kW, 150kW, -300kW, -200kW, 200kW
   * Consumption Offers: 150kW, -400kW, 50kW, -100kW
   * Solveable: yes
   * @return The constructed problem.
   */
  public static ProductionConsumptionBalancing createScenario5() {

    ProductionConsumptionBalancing problem = null;

    final String entity = "id";

    // Problem facts
    long totalProduction = 1000000; // [1000 kW]
    long totalConsumption = 750000; // [750 kW]

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();

    // Consumption offers: [150 kW], [-400 kW], [-50 kW], [-100 kW]
    events.add(new ConsumptionOfferEvent(0, entity, 150000, new
        NonThrottledEvent(0, entity)));
    events.add(new ConsumptionOfferEvent(1, entity, -400000, new
        NonThrottledEvent(1, entity)));
    events.add(new ConsumptionOfferEvent(2, entity, -50000, new
        NonThrottledEvent(2, entity)));
    events.add(new ConsumptionOfferEvent(3, entity, -100000, new
        NonThrottledEvent(3, entity)));



    // Production offers: [-150 kW], [50 kW], [150 kW], [-300 kW], [-200 kW],
        [200 kW]
```

```
    events.add(new ProductionOfferEvent(4, entity, -150000, new
        NonThrottledEvent(4, entity)));
    events.add(new ProductionOfferEvent(5, entity, 50000, new
        NonThrottledEvent(5, entity)));
    events.add(new ProductionOfferEvent(6, entity, 150000, new
        NonThrottledEvent(6, entity)));
    events.add(new ProductionOfferEvent(7, entity, -300000, new
        NonThrottledEvent(7, entity)));
    events.add(new ProductionOfferEvent(8, entity, -200000, new
        NonThrottledEvent(8, entity)));
    events.add(new ProductionOfferEvent(9, entity, 200000, new
        NonThrottledEvent(9, entity)));

    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);

    return problem;

  }

  /**
   * Scenario initial balancing Problem:
   * Production: 0kW
   * Consumption: 0kW
   * Prodction Offers: -
   * Consumption Offers: -
   * Solveable: yes
   * @return The constructed problem.
   */
  public static ProductionConsumptionBalancing createEmptyBalancingProblem() {

    ProductionConsumptionBalancing problem = null;

    final long totalProduction = 0L;
    final long totalConsumption = 0L;

    List<BaseOfferEvent> events = new ArrayList<BaseOfferEvent>();

    // Create unsolved problem
    problem = ProductionConsumptionBalancingGenerator
          .CreateBalancingProblem(totalProduction, totalConsumption, events);
```

```
      return problem;

   }
}
```

Listing C.1: Implementierung der Szenarien für die Evaluierung von Planungsalgorithmen

D. Implementierung der ScheduleItems

```
@PlanningEntity
public class ScheduleItem implements Cloneable{

   private BaseOfferEvent item;
   private int identifier;

   public ScheduleItem() {
      this(0);
   }

   public ScheduleItem(int id) {
      this.item = null;
      this.identifier = id;
   }

   @PlanningVariable(strengthComparatorClass = ChangeOfferEventComperator.class)
   @ValueRanges({
         @ValueRange(type = ValueRangeType.FROM_SOLUTION_PROPERTY, solutionProperty
             = "dummies"),
         @ValueRange(type = ValueRangeType.FROM_SOLUTION_PROPERTY, solutionProperty
             = "offers")


   })
   public BaseOfferEvent getItem() {
      return item;
   }

   public void setItem(BaseOfferEvent item) {
      this.item = item;
   }

   public int getIdentifier() {
      return identifier;
   }
```

```
   @Override
   public boolean equals(Object obj) {

      if (obj == null) {
         return false;
      }

      if (obj == this) {
         return true;
      }

      if (obj.getClass() != getClass()) {
         return false;
      }

      ScheduleItem other = (ScheduleItem) obj;

      return new EqualsBuilder()
         .append(this.item, other.item)
         .append(this.getIdentifier(), other.getIdentifier())
         .isEquals();
   }

   @Override
   public int hashCode() {
//    return new HashCodeBuilder(3, 5)
//       .append(this.item.hashCode())
//       .toHashCode();

      HashCodeBuilder hb = new HashCodeBuilder(3, 5);

      if(this.item != null) {
         hb.append(this.item);
      }

      hb.append(this.identifier);

      return hb.toHashCode();
   }
```

```
   @Override
   public String toString() {
      StringBuilder sb = new StringBuilder();

      sb.append("[ScheduleItem id:");
      sb.append(this.identifier);

      sb.append(" item:");
      if(item != null) {
         sb.append(item.toString());
      } else {
         sb.append("null");
      }

      sb.append("]");

      return sb.toString();
   }

   public Object clone() {

      ScheduleItem clone = null;

      try {

           clone = (ScheduleItem) super.clone();

        }
        catch (CloneNotSupportedException e) {
            // This should never happen
            throw new InternalError(e.toString());
        }


      clone.item = this.item;

      clone.identifier = this.identifier;

      return clone;
   }

}
```

Listing D.1: Implementierung der Klasse ScheduleItem

E. Ereignisverifikation mittels AutoResetEvent

```
[TestMethod]
 public void SendAndReceiveTest()
 {
     MI.MessagingAdapter msgAdapter = new MI.MqttMessagingAdapter(BROKER_ADR,
         BROKER_PORT, CLIENT_ID);
     int msgReceived = 0;
     int msgSend = 10;
     string testTopic = "test";

     msgAdapter.Connect();

     MI.IMessageNotificationService receiver =
         (MI.IMessageNotificationService)msgAdapter;
     MI.IMessagePublishingService sender =
         (MI.IMessagePublishingService)msgAdapter;

     AutoResetEvent waitOnReceivedMessages = new AutoResetEvent(false);

     MI.IMessageNotifyee testNotifyee = new DelegateNotifyee((topic, msg) =>
     {
         msgReceived++;
         if (msgReceived >= msgSend)
         {
             // Send ResetEvent signaling that all messages have been received
             waitOnReceivedMessages.Set();
         }
     });

     logger.Debug("Registering for test topic.");

     // Register Notifyee for messages
     receiver.Subscribe(testNotifyee, testTopic);

     logger.Debug("For test topic registered.");
```

```
        logger.Debug("Sending messages ...");

        if (((MI.MqttMessagingAdapter)msgAdapter).IsConnected)
        {
            for (int i = 1; i <= msgSend; i++)
            {
                sender.PublishMessageToTopic("Testmessage: " + i, testTopic);
            }
        }

        logger.Debug(msgSend + "messages have been send.");

        // Wait 10 seconds for sending and receiving all messages
        waitOnReceivedMessages.WaitOne(10000);

        // Check if all messages were received
        Assert.AreEqual(msgSend, msgReceived);

        // Unsubscribe from topic
        receiver.Unsubscribe(testNotifyee, testTopic);

        // Disconnect from broker
        msgAdapter.Disconnect();
    }
```

Listing E.1: Verifikation eines Ereignisses mittels AutoResetEvent

F. Inhalt der CD-ROM

Dieser Arbeit liegt eine CD-ROM mit folgender Verzeichnisstruktur bei:

- **[Ausarbeitung]** enthält die Arbeit im PDF-Format.
- **[Quellcode]** enthält die Projektordner der einzelnen Teilsysteme mit dem dazugehörigen Quellcode und den benötigten Bibliotheken.

Literaturverzeichnis

[Armstrong 2007] Armstrong, Joe: *Programming Erlang: Software for a Concurrent World*. Pragmatic Bookshelf, 2007. – ISBN 193435600X, 9781934356005

[Blume 2008] Blume, S.W.: *Electric Power System Basics for the Nonelectrical Professional*. Wiley, 2008 (IEEE Press Series on Power Engineering). – URL http://books.google.de/books?id=_mXaEA986xIC. – ISBN 9780470185803

[Bundesamt für Bevölkerungsschutz und Katastrophenhilfe 2011] Bundesamt für Bevölkerungsschutz und Katastrophenhilfe: *Krisenmanagement Stromausfall*. 2011. – URL http://www.bbk.bund.de/SharedDocs/Downloads/BBK/DE/Publikationen/PublikationenKritis/Krisenhandbuch_Stromausfall_Kurzfassung_pdf.pdf?__blob=publicationFile

[Bundesanstalt für Geowissenschaften und Rohstoffe 2012] Bundesanstalt für Geowissenschaften und Rohstoffe: *Energiestudie 2012 - Reserven, Ressourcen und Verfügbarkeit von Energierohstoffen*. Dezember 2012. – URL http://www.bgr.bund.de/DE/Gemeinsames/Produkte/Downloads/DERA_Rohstoffinformationen/rohstoffinformationen-15.pdf?__blob=publicationFile&v=6

[Bundesnetzagentur 2012] Bundesnetzagentur: Bericht zum Zustand der leitungsgebundenen Energieversorgung im Winter 2011/12. URL http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/BNetzA/Presse/Berichte/2012/NetzBericht_ZustandWinter11_12pdf.pdf?__blob=publicationFile, Mai 2012. – Forschungsbericht

[Chassin u. a. 2008] Chassin, D.P. ; Schneider, K. ; Gerkensmeyer, C.: GridLAB-D: An open-source power systems modeling and simulation environment. In: *Transmission and Distribution Conference and Exposition, 2008. T D. IEEE/PES*, 2008, S. 1–5

[Chassin und Widergren 2009] Chassin, D.P. ; Widergren, S.E.: Simulating demand participation in market operations. In: *Power Energy Society General Meeting, 2009. PES '09. IEEE*, 2009, S. 1–5. – ISSN 1944-9925

[Croft u. a. 2011] Croft, A. ; Madawala, U.K. ; Thrimawithana, D.J.: Simulation platform for micro-grids with demand-side management. In: *IECON 2011 - 37th Annual Conference on IEEE Industrial Electronics Society*, nov. 2011, S. 3254 –3259. – ISSN 1553-572X

[Deutsche Energie-Agentur GmbH 2012] Deutsche Energie-Agentur GmbH: *Das deutsche Stromnetz*. 2012. – URL http://www.dena.de/fileadmin/user_upload/Presse/Meldungen/2012/Schaubild_Stromnetz.pdf

[El-Hawary 2008] El-Hawary, M.E.: *Introduction to Electrical Power Systems*. Wiley, 2008 (IEEE Press Series on Power Engineering). – URL http://books.google.de/books?id=Lo3b0OQsy0MC. – ISBN 9780470411360

[E.ON Mitte AG 2013] E.ON Mitte AG: *Standardlastprofilverfahren*, 2013. – URL http://www.eon-mitte.com/admin/userimages/File/netz2008/Vertraege/Strom/NNV_Variante_2/080901_NNV_Variante_2_Anlage_3.pdf

[Etzion und Niblett 2010] Etzion, Opher ; Niblett, Peter: *Event Processing in Action*. 1st. Greenwich, CT, USA : Manning Publications Co., 2010. – ISBN 1935182218, 9781935182214

[Farhangi 2010] Farhangi, H.: The path of the smart grid. In: *Power and Energy Magazine, IEEE* 8 (2010), Nr. 1, S. 18–28. – ISSN 1540-7977

[Fowler 2002] Fowler, Martin: *Patterns of Enterprise Application Architecture*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. – ISBN 0321127420

[Gellings 1985] Gellings, C.W.: The concept of demand-side management for electric utilities. In: *Proceedings of the IEEE* 73 (1985), Nr. 10, S. 1468–1470. – ISSN 0018-9219

[Givler und Lilienthal 2005] Givler, T. ; Lilienthal, P.: Using HOMER Software, NREL's Micropower Optimization Model, to Explore the Role of Gen-sets in Small Solar Power Systems / National Renewable Energy Laboratory. URL http://www.nrel.gov/docs/fy05osti/36774.pdf, 2005. – Forschungsbericht

[Grehn u. a. 1998] Grehn, J. ; Bolz, J. ; Krause, J.: *Metzler Physik*. Bd. 3: *Metzler Physik*. Schroedel Verlag GmbH, 1998. – URL http://books.google.de/books?id=rFmCAAAACAAJ. – ISBN 9783507107106

[Gudi u. a. 2011] Gudi, N. ; Wang, Lingfeng ; Devabhaktuni, V. ; Depuru, S.S.S.R.: A demand-side management simulation platform incorporating optimal management of distributed renewable resources. In: *Power Systems Conference and Exposition (PSCE), 2011 IEEE/PES*, march 2011, S. 1 –7

[Gungor u. a. 2011] Gungor, V.C. ; Sahin, D. ; Kocak, T. ; Ergut, S. ; Buccella, C. ; Cecati, C. ; Hancke, G.P.: Smart Grid Technologies: Communication Technologies and Standards. In: *Industrial Informatics, IEEE Transactions on* 7 (2011), Nr. 4, S. 529–539. – ISSN 1551-3203

[Hassan und Radman 2010] Hassan, R. ; Radman, G.: Survey on Smart Grid. In: *IEEE SoutheastCon 2010 (SoutheastCon), Proceedings of the*, 2010, S. 210–213

[Hatziargyriou u. a. 2007] Hatziargyriou, N. ; Asano, H. ; Iravani, R. ; Marnay, C.: Microgrids. In: *Power and Energy Magazine, IEEE* 5 (2007), Nr. 4, S. 78–94. – ISSN 1540-7977

[Haubrich 2008] Haubrich, Hans-Jürgen: *Gutachten zur Höhe des Regelenergiebedarfs*. 2008. – URL http://www.bundesnetzagentur.de/cae/servlet/contentblob/102556/publicationFile/5861/Gutachten%20zur%20H%C3%B6he%20des%20Regelenergiebedarfes.pdf

[Hewitt 2010] Hewitt, Carl: Actor Model for Discretionary, Adaptive Concurrency. In: *CoRR* abs/1008.1459 (2010)

[Hewitt u. a. 1973] Hewitt, Carl ; Bishop, Peter ; Steiger, Richard: A universal modular ACTOR formalism for artificial intelligence. In: *Proceedings of the 3rd international joint conference on Artificial intelligence*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1973 (IJCAI'73), S. 235–245. – URL http://dl.acm.org/citation.cfm?id=1624775.1624804

[Huang u. a. 2011] Huang, Wei ; Lu, Miao ; Zhang, Li: Survey on Microgrid Control Strategies. In: *Energy Procedia* 12 (2011), Nr. 0, S. 206 – 212. – URL http://www.sciencedirect.com/science/article/pii/S1876610211018558. – <ce:title>The Proceedings of International Conference on Smart Grid and Clean Energy Technologies (ICSGCE 2011</ce:title>. – ISSN 1876-6102

[International Energy Agency 2012] International Energy Agency: *World Energy Outlook 2012*. 2012

[Kiliccote u. a. 2006] Kiliccote, Sila ; Piette, Mary A. ; Hansen, David: Advanced Controls and Communications for Demand Response and Energy Efficiency in Commercial Buildings. In: *Second Carnegie Mellon Conference in Electric Power Systems: Monitoring, Sensing, Software and Its Valuation for the Changing Electric Power Industry*, Januar 2006

[Klobasa 2006] Klobasa, Marian: *Demand Side Management in dezentral geführten Verteilnetzen*. November 2006. – URL http://www.iset.uni-kassel.de/KSES/2006/PDF/Session_3/Klobasa-KSES-2006.pdf

[Kundur u. a. 1994] Kundur, P. ; Balu, N.J. ; Lauby, M.G.: Power System Stability and Control. (1994). – URL http://books.google.de/books?id=2cbvyf8Ly4AC. ISBN 9780070359581

[Kupzog 2006] Kupzog, F.: Self-controlled Exploitation of Energy Cost saving Potentials by Implementing Distributed Demand Side Management. In: *Industrial Informatics, 2006 IEEE International Conference on*, 2006, S. 375–380

[Kupzog 2008] Kupzog, Friederich: *Frequency-Responsive Load Management in ElectricPower Grids: A Technical Concept for Demand Response in Smart Grids*. Südwestdeutscher Verlag, 2008. – URL http://books.google.de/books?id=Nk7tPAAACAAJ. – ISBN 9783838101255

[Lambert u. a. 2006] Lambert, T. ; Gilman, P. ; Lilienthal, P.: *Micropower system modeling with HOMER*. Kap. 15, S. 379 – 418, John Wiley & Sons, 2006. – URL `http://www.pspb.org/e21/media/HOMERModelingInformation.pdf`

[Lilienthal u. a. 2005] Lilienthal, P. ; Lambert, T. ; Gilman, P.: *Getting started guide for homer version 2.1.* National Renewable Energy Laboratory (Veranst.), 2005. – URL `https://analysis.nrel.gov/homer/includes/downloads/HOMERGettingStarted210.pdf`

[Luckham und Schulte 2013] Luckham, David ; Schulte, W. R.: Why Companies Should Develop Event Models. (2013), Januar. – URL `http://www.complexevents.com/wp-content/uploads/2013/01/Event-Models-Luckham-schulte-Final-Jan-2013.pdf`

[Luckham 2001] Luckham, David C.: *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems.* Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2001. – ISBN 0201727897

[Mensk und Gardemann 2008] Mensk, Ute ; Gardemann, Joachim: Auswirkungen des Ausfalls kritischer Infrastrukturen auf den Ernährungssektor am Beispiel des Stromausfalls im Münsterland im Herbst 2005 / Fachhochschule Münster. URL `http://www.hb.fh-muenster.de/opus/fhms/volltexte/2011/677/pdf/Stromausfall_Muensterland.pdf`, 2008. – Forschungsbericht

[Odersky u. a. 2008] Odersky, Martin ; Spoon, Lex ; Venners, Bill: *Programming in Scala: A Comprehensive Step-by-step Guide.* 1st. USA : Artima Incorporation, 2008. – ISBN 0981531601, 9780981531601

[Serafin von Roon 2010] Roon, Malte H. Serafin von: *Demand Side Management in Haushalten.* April 2010. – URL `http://www.ffe.de/download/wissen/20100406_Methodik_DSM.pdf`

[Schneider u. a. 2009] Schneider, K.P. ; Chassin, D. ; Chen, Y. ; Fuller, J.C.: Distribution power flow for smart grid technologies. In: *Power Systems Conference and Exposition, 2009. PSCE '09. IEEE/PES*, 2009, S. 1–7

[Schommers 2011] Schommers, A.: *Elektronik - gar nicht schwer.* Bd. 11: *Elektronik - gar nicht schwer: Entdecken, probieren, verstehen : Buch 1 : Experimente mit Gleichstrom.* Elektor-Verlag, 2011. – URL `http://books.google.de/books?id=CzkbSAAACAAJ`. – ISBN 9783921608326

[Short u. a. 2007] Short, J.A. ; Infield, D.G. ; Freris, L.L.: Stabilization of Grid Frequency Through Dynamic Demand Control. In: *Power Systems, IEEE Transactions on* 22 (2007), aug., Nr. 3, S. 1284 –1293. – ISSN 0885-8950

[Siedersleben 2004] Siedersleben, Johannes (Hrsg.): *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar.* Heidelberg : dpunkt, 2004. – ISBN 978-3-89864-292-7

[Steudte 2011] Steudte, Armin: *Konzeption und Entwicklung eines ereignisgesteuerten Frameworks für Ambient Assisted Living Anwendungen.* Mai 2011. – URL `http://opus.haw-hamburg.de/volltexte/2011/1337/pdf/steudte_thesis.pdf`

[Steudte 2013] Steudte, Armin: *Projektbericht 2 - Entwurf und Implementierung einer Experimentierumgebung für Demand Side Management.* März 2013

[Stimoniaris u. a. 2011] Stimoniaris, D. ; Tsiamitros, D. ; Kottas, T. ; Asimopoulos, N. ; Dialynas, E.: Smart grid simulation using small-scale pilot installations. - experimental investigation of a centrally-controlled microgrid. In: *PowerTech, 2011 IEEE Trondheim*, IEEE, 2011, S. 1–6

[Strbac 2008] Strbac, Goran: Demand side management: Benefits and challenges. In: *Energy Policy* 36 (2008), Nr. 12, S. 4419–4426

[Sven Teske 2007] Sven Teske, Oliver S.: *Energy Revolution.* Januar 2007. – URL `http://www.energyblueprint.info/fileadmin/media/documents/energy_revolution.pdf`

[Zaidi u. a. 2010] Zaidi, A.A. ; Zia, T. ; Kupzog, F.: Automated demand side management in microgrids using load recognition. In: *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, july 2010, S. 774 –779

[Zeilinger 2011] Zeilinger, F.: Simulation of the effect of demand side management to the power consumption of households. In: *Energetics (IYCE), Proceedings of the 2011 3rd International Youth Conference on*, july 2011, S. 1 –9

Abkürzungsverzeichnis

DSM Demand Side Management

CEP Complex Event Processing

EDA Event-Driven Architecture

MOM Message-Oriented Middleware

CGMU Central Grid Management Unit

HGMU Home Grid Management Unit

MVVM Model-View-ViewModel-Entwurfsmuster

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 26.08.2013 Armin Steudte