CARL VON OSSIETZKY universität OLDENBURG

Department für Informatik – Abteilung Umweltinformatik

Abschlussbericht

Sommersemester 2014 & Wintersemester 2014/'15

Vorwort

Dieser Abschlussbericht ist im Rahmen der Projektgruppe *Hardwarebasierte Simulation energieautonomer Gebäude* an der Carl von Ossietzky Universität Oldenburg, Department für Informatik, Abteilung Umweltinformatik entstanden.

Oldenburg,
März 2015

Inhaltsverzeichnis

Abbildungsverzeichnis

Betreuer und Teilnehmer

Prof. Dr. Michael Sonnenschein (Betreuer)
sonnenschein@informatik.uni-oldenburg.de

Dr. Ute Vogel (Betreuer)
vogel@informatik.uni-oldenburg.de

M.Sc. Christian Hinrichs (Betreuer)
hinrichs@informatik.uni-oldenburg.de

Tobias Kölker
tobias.koelker@uni-oldenburg.de

Marius Hacker
marius.hacker@uni-oldenburg.de

Michael Cordes
michael.cordes@uni-oldenburg.de

Dag Ennenga
dag.ennenga@uni-oldenburg.de

Eugen Langolf
eugen.langolf@uni-oldenburg.de

Connor Fibich
connor.fibich@uni-oldenburg.de

Kevin Möhlmann
kevin.moehlmann@uni-oldenburg.de

Hendrik Grunau
hendrik.grunau@uni-oldenburg.de

Sönke Martens
soenke.martens@uni-oldenburg.de

Claas Martin
claas.martin@uni-oldenburg.de

Christian Best
christian.best@uni-oldenburg.de

Abkürzungsverzeichnis

ADS	Automation Device Specification. 107, 179
CEM	Customer Energy Manager. 121, 164, 170
DB	Datenbank. 138
DBMS	Datenbankmanagementsystem. 138, 164
DWD	Deutscher Wetterdienst. 147, 196, 248
EMS	Energiemanagementsystem. 21, 41–45, 47, 53–57, 63, 64, 67, 70, 73–75, 77, 78, 80, 83, 125–127, 129, 131, 135, 137, 138, 145, 154, 156, 158–160, 192, 195, 196, 198, 203–205, 207, 209, 213
EVS	Energieversorgungssystem. 43
GUI	grafische Benutzeroberfläche. 19, 192, 195, 196
ID	Identifikator. 97
KWK	Kraft-Wärme-Kopplung. 1
OPC-UA	OPC Unified Architecture. 106
SE	Steuerungsebene. 21, 47, 57, 125, 127, 155, 158, 192
SF	Simulationsframework. 21, 41–44, 47, 57, 104, 125–127, 129, 137, 156, 158, 159, 192, 195, 196
UC	Use-Case (deutsch: Anwendungsfall). 43

Kapitel 1
Einleitung

In den letzten Jahren ist die Anzahl der dezentralen Energieerzeugungsanlage stetig gestiegen [43], da der Erwerb und Betrieb von dezentralen Energieanlagen durch die Subventionierung erneuerbaren Energien, den technischen Fortschritt [63] und steigender Preise fossiler Energieträger lukrativer geworden ist [5]. Der steigende Ausbau der dezentralen Energieanlagen bedeutet jedoch eine zusätzliche Belastung für das Energienetz, denn das Netz ist zum einen nicht für eine große Einspeisung auf der Nieder- und Mittelspannung ausgelegt [21] und zum anderen erschwert die zum Teil stark fluktuierende Einspeisung der erneuerbaren Energien das Gleichgewicht von Verbrauch und Erzeugung [33].

Eine Möglichkeit diese negativen Effekte teilweise zu kompensieren ist den erzeugten Strom dort zu verwenden, wo er entstanden ist. Das entlastet nicht nur die Netze, sondern ermöglicht es dem Betreiber einer dezentralen Anlage, seine Energiebezugskosten durch den Eigenverbrauch des Stroms zu senken. Die Maximierung des Eigenverbrauchs ist auf ersten Blick nicht möglich, weil dezentrale Energieanlagen, wie Photovoltaik-, Windenergie- oder KWK-Anlagen nicht oder nur teilweise steuerbar sind, sondern von Sonneneinstrahlung, Windstärke bzw. Wärmebedarf abhängen. Um dennoch einen hohen Eigenverbrauch bzw. minimale Energiebezugskosten für den Betreiber zu ermöglichen, ist es unser Ziel, ein Energiemanagementsystem zu entwickeln, dass den Einsatz steuerbarer Verbraucher und Energiespeicher mit dem Ziel der Energiebezugskostenminimierung optimiert. Neben den Energiemanagementsystem bietet unser Projekt ein Simulationsframework zum Testen und Evaluieren der Optimierungsmethode.

Dieser Abschlussbericht stellt dabei die Planung, Durchführung und Ergebnisse unseres Projekts vor. Dazu werden als Grundlage unseres Vorgehens im nächsten Abschnitt die Ziele des Projekts erläutert. Anschließend wird in Abschnitt 1.2 ein Überblick über den Aufbau des Abschlussberichts geboten.

1.1 Vision

Die Projektgruppe „Hardware-basierte Simulation energieautonomer Gebäude“ verfolgt die Vision, ein Energiemanagementsystem für ein aus mehreren Komponenten bestehendes Simulationssystem zu entwickeln, das eine Eigenverbrauchsoptimierung durchführt. Die Komponenten bestehen aus Energieerzeugern, -speichern und -verbrauchern als Teil des simulierten Gebäudes. Die Energieerzeuger zur Versorgung des Gebäudes mit elektrischer Energie sind Photovoltaik-Anlagen und/oder Blockheizkraftwerke, die zusätzlich thermische Energie erzeugen. Als Speicher stehen Energiespeicher und thermische Speicher, sogenannte Pufferspeicher, zur Verfügung. Die Verbraucher werden unterteilt in nicht-steuerbare Verbraucher, die die Grundlast bilden, und steuerbare Verbraucher. Bei dem Gebäude soll es sich weder um einen Industriebetrieb, noch um einen Privathaushalt handeln, sondern allgemein um ein großes Gebäude mit gegebenen Wärmebedarf, hohem Stromverbrauch und steuerbaren Verbrauchern innerhalb des Gebäudes. In Frage kommt unter Anderem die Simulation eines Krankenhauses, einer Schule oder eines Hotels.

Für die hardware-basierte Simulation soll von der Projektgruppe ein Simulationssystem erstellt werden, das für beliebige Szenarien mit unterschiedlichen Rahmenbedingungen die Eigenverbrauchsoptimierung durchführt. Es sollen unterschiedliche Gebäude hinsichtlich des Zwecks und der Größe unter frei wählbaren Wetter- und Klimabedingungen analysierbar sein. Auch sollen weitere Energieverbraucher innerhalb des Gebäudes manuell hinzugefügt oder entfernt werden können. Ein solches Szenario beinhaltet somit die Kenndaten des definierten Gebäudes, also Grundlast bezüglich Strom- und Wärmebedarf, Eigenschaften über die steuerbaren Verbraucher und die dezentralen Energieerzeuger und -speicher in der Simulation. Dazu gehört unter anderem die Dimensionierung der Energieanlagen, das heißt Angaben darüber, wie viel elektrische und thermische Energie durch die Anlagen bereitgestellt werden können. Des Weiteren werden zwei Wetterdatenquellen für das Szenario verwendet. Die erste besteht aus Vorhersagedaten, welche zur Berechnung von Einspeisedaten und Lastprognosen genutzt werden. Die zweite Quelle liefert reelle Wetterdaten, die von den Vorhersagen abweichen können.

Mit dem Start der Simulation soll durch das entwickelte Energiemanagementsystem ein Ablaufplan, der sogenannte „Fahrplan“, für die Energieversorgung berechnet werden, der in 15 minütiger Taktung in Abhängigkeit von veränderten Last- bzw. Wetterdaten aktualisiert/angepasst wird. Ziel der Fahrplanerstellung ist die Eigenverbrauchsoptimierung zur Minimierung der Energiebezugskosten im Rahmen der definierten Simulationsdauer. Das Ergebnis soll hinsichtlich der Fragen evaluiert werden können, was das Energiemanagement im Bezug auf ein Szenario bewirkt habe, ob dem öffentlichen Stromnetz mehr Energie im Vergleich zum ungesteuerten Ablaufplan entnommen werden musste oder welche Parameter beziehungsweise Konfiguration im Vergleich verschiedener Szenarien einen positiven Einfluss auf das erzielte Ergebnis hatte.

1.2 Aufbau des Dokuments

Dieser Bericht beschreibt das Vorgehen, die Konzepte und deren Umsetzungen, um die Ziele der Vision zu erreichen. Die einzelnen Kapitel des Dokuments beschreiben wichtige Eckpunkte unseres Vorgehens und unserer Lösungen. Das Dokument beginnt in Abschnitt 2 mit einer Beschreibung des Vorgehens und den Organisationsstrukturen. Unter anderem werden hier unser Vorgehensmodell, die Rollenbeschreibungen der Projektteilnehmer und der Projektablauf beschrieben. Anschließend folgt die Problemanalyse in Abschnitt 3, in welcher das zu lösende Problem in einzelne Akteure, Komponenten und Anwendungsfälle aufgeteilt wird. Darauf aufbauend wurden Anforderungen ermittelt und in Abschnitt 4 in Kategorien aufgeteilt und detailliert spezifiziert. Unter Berücksichtigung der Anforderungen wurden Konzepte entwickelt und im Entwurf in Abschnitt 5 verfasst. Diese Konzepte umfassen unter Anderem unseren Hard- und Softwareentwurf, das Datenbankkonzept, die Wahl des Kommunikationsprotokolls sowie die Grundlagen der Optimierung. Der Entwurf dient dabei auch als Grundlage für die konkreten Realisierungen in Abschnitt 6, in dem die wichtigsten Implementierungsdetails und Umsetzungen beschrieben werden. Hiernach werden in Abschnitt 7 die Fähigkeiten der einzelnen Softwareversionen beleuchtet. In Abschnitt 8 wird untersucht, wie die Anforderungen erfüllt wurden. Daran anschließend ist mit Abschnitt 9 eine Evaluation des Energiemanagementsystems gegeben, welche zunächst einige Grundlagen der Untersuchung klärt bevor eine Auswertung präsentiert wird. Abschließend werden in Abschnitt 10 die vergangenen zwei Semester reflektiert und analysiert und der Bericht mit einem Ausblick auf mögliche Erweiterungen des Softwaresystems abgeschlossen.

Kapitel 2
Vorgehen

Im folgenden Kapitel wird das Vorgehen und die Organisationsstruktur des Projekts beschrieben. Unter anderem wurden die Teilnehmer des Projekts in unterschiedlichen Rollen eingeteilt, die beschreiben welches Teilgebiet des Projekts ihrer Verantwortung obliegt. Die Rolleneinteilung und einzelnen Rollen sind in Abschnitt 2.1 beschrieben.

Um eine erfolgreiche Bearbeitung des Projekts zu gewährleisten, haben wir uns dazu entschlossen ein Vorgehensmodell zu benutzen, welches einen Rahmen für die zeitliche Einteilung des Projekts und Erstellung von Aufgabenpaketen bietet. Die Wahl und Beschreibung des Vorgehensmodells befindet sich in Abschnitt 2.2. Aufbauend auf das Vorgehensmodell werden im Projektablauf in Abschnitt 2.3 die bisher durchgeführten Iterationen beschrieben.

Anschließend werden in weiteren Abschnitten wichtige Teilaspekte unserer Organisation dargelegt. In Abschnitt 2.4 wird beschrieben, welche Software wir einsetzen, um uns bei der Organisation des Projekts zu unterstützen. Weiterhin werden in Abschnitt 2.5 und Abschnitt 2.6 das Qualitäts- und Testmanagement erläutert, welche die Qualität des Projekts und die Korrektheit der Software sicherstellen sollen.

2.1 Rollen

Für eine funktionierende Organisation ist es sinnvoll, für bedeutende Aufgabenbereiche Ansprechpartner (*Rollen*) zu benennen. Das Übernehmen einer Rolle geht einher mit der Verantwortung für den definierten Bereich und - sofern notwendig - der Zuweisung von technischen Kompetenzen. Ferner sorgen Rollen für verkürzte Kommunikationswege, indem sich Gruppenmitglieder sowie Betreuer an entsprechende Ansprechpartner wenden können. Bei hohem Arbeitsaufwand ziehen diese weitere Personen hinzu oder setzen sich mit dem Projektmanagement in Verbindung. Für Urlaubs- und Krankheitsfälle sind Vertreter benannt.

Die für unser Projekt wesentlichen Rollen werden im Folgenden kurz durch eine Rollenbeschreibung dargestellt, die verbindlich die Aufgabenbereiche regelt.

Um einen Einblick in neue Bereiche zu kommen und gleichzeitig Inselwissen im Sinne des Risikomanagements zu reduzieren, wurden nach einem Semester die Rollen gewechselt. Bei den Rollen wird jeweils erläutert, wer in welchem Semester die jeweilige Rolle inne hatte.

2.1.1 Qualitäts-/Testmanager

SS 2014 Rolleninhaber: Christian Best, Stellvertreter: Tobias Koelker
WS 2014/'15 Rolleninhaber: Tobias Koelker & Eugen Langolf, Stellvertreter: Claas Martin & Christian Best

Der Qualitätsmanager gilt für die Projektgruppe als interner Dienstleister und Berater für das Qualitätsmanagement. Die Aufgabe des Qualitätsmanagers besteht in der Anfangsphase des Projekts in der Einführung des Qualitätsmanagements. Dazu gehört das Anlegen einer Confluencestruktur, in der Regeln und Vereinbarung für das gesamte Team dokumentiert werden können. Während der Anforderungsanalyse unterstützt er das Anforderungsmanagement bei der ordnungsgemäßen Dokumentation von Anforderungen. Im gesamten Verlauf ist der Qualitätsmanager für die Einhaltung der getroffenen Vereinbarung und die Weiterentwicklung des Qualitätsmanagements verantwortlich. Er kontrolliert die Fertigstellung und Qualität von Protokollen und extrahiert daraus Aufgaben, Arbeitspakete und Vereinbarungen, die wiederum im Confluence dokumentiert werden. Für erstellte Dokumente organisiert der Qualitätsmanager zur Gewährleistung der Qualität Review-Termine, an denen sowohl der Autor des Dokuments als auch weitere Projektmitglieder teilnehmen. Weiterhin verantwortet er die Inspektionen von geschriebenen Codezeilen, die den definierten Standards entsprechen müssen. Allgemein soll der Qualitätsmanager zu jeder Zeit als Ansprechpartner für Fragen, die die Qualität betreffen, zur Verfügung stehen.

Der Testmanager erarbeitet ein Testkonzept für die Artefakte des Entwicklungsprozesses. Dazu zählen sowohl manuelle, wie auch automatische Tests. Diese tragen dafür Sorge, dass die Artefakte ihr in den Anforderungen gefordertes Verhalten erfüllen und somit eine hohe Qualität haben. Insbesondere wichtig sind Regressionstests, um zu verhindern, dass sich bei späteren Änderungen in bereits den Anforderungen entsprechenden Implementierungen Fehler einschleichen. Es soll ein Testkonzepts in Zusammenarbeit mit den Architekten erstellt werden um die Testbarkeit des Systems zu vereinfachen.

Schnittstellen

Der Qualitätsmanager und auch der Testmanager stimmen ihre Arbeit mit der Projektleitung ab, um zu planen, welche Kapazitäten für das Qualitäts- / Testmanagement notwendig sind. Der Qualitätsmanager bietet allen anderen Rollen seine Beratung zur Qualitätsverbesserung an. Er empfiehlt Maßnahmen zur Qualitätssicherung und unterstützt bei deren Umsetzung. Der Testmanager arbeitet eng mit den Architekten zusammen, um die Testbarkeit des Systems zu ermöglichen und vereinfachen. Zudem erarbeiten Architekten und Testmanager gemeinsam ein Testkonzept. Der Testmanager berät und unterstützt die Entwickler bei der Umsetzung des Testkonzepts. Da Tests eine Maßnahme zur Sicherstellung der Qualität sind, stimmt der Testmanager diese Maßnahmen mit dem Qualitätsmanager ab. Darüber hinaus erfolgt eine enge Zusammenarbeit mit dem Infrastrukturbeauftragten, um eine möglichst einfache Sicherstellung der Qualität zu ermöglichen, die sich in die Arbeitsläufe der Entwickler integrieren. Es erfolgt kontinuierliche Integration, wie in Abschnitt 2.4 erläutert, zur Steigerung der Softwarequalität.

2.1.2 Dokumentenbeauftragter

SS 2014 Rolleninhaber: Sönke Martens, Stellvertreter: Michael Cordes
WS 2014/'15 Rolleninhaber: Hendrik Grunau, Stellvertreter: Dag Ennenga

Der Dokumentenbeauftragte ist für die Verwaltung und Strukturierung der Dokumente zuständig, die im Laufe des Projekts entstehen. Zu Beginn des Projekts ist er verantwortlich für die Erhebung von Anforderungen an die Dokumentation. Aus diesen leitet er Dokumenten-Templates ab, die bei der Erstellung der Dokumente als Grundlage dienen. Diese Dokumenten-Templates sollen mit Hilfe einer Gliederung und Beispielen zu einzelnen Abschnitten, eine einheitliche Struktur ermöglichen. Darüber hinaus fällt am Ende des Projektes die Planung des Abschlussberichts und des Benutzerhandbuchs in den Aufgabenbereich. Dafür müssen die Aufgabenpakete geplant und verteilt werden. Weiterhin ist er für die Qualität der schriftlichen Dokumente zuständig. Dazu leitet er Maßnahmen ein, welche die Qualität der Dokumente sicherstellen, wie das Überprüfen der Dokumente durch Reviews. Er ist außerdem der Ansprechpartner bei Fragen bezüglich der Dokumentation.

Schnittstellen

Der Dokumentenbeauftragte erstellt in der Anfangsphase ein Template für das Anforderungsdokument, das er mit dem Anforderungsmanagment abstimmt. Des weiteren berät er sich mit dem Projektmanagement, so dass die Dokumente mit dem Prozessmodell des Projekts vereinbar sind. Außerdem bietet er Templates für die Entwickler des Teams, in denen diese Projektfortschritte dokumentieren. Welche Teile genau

dokumentiert werden müssen, bespricht er mit den jeweils zuständigen Teammitgliedern.

2.1.3 Infrastrukturbeauftragter

SS 2014 Rolleninhaber: Michael Cordes, Stellvertreter: Christian Best
WS 2014/'15 Rolleninhaber: Marius Hacker, Stellverter: Michael Cordes

Der Infrastrukturbeauftragte ist für die Softwarekomponente der Infrastruktur verantwortlich. Er ist dafür zuständig, Repositories aufzusetzen und falls notwendig zerstörte Repositories soweit wie möglich wieder herzustellen. Ggfs. müssen Branches angelegt werden und die Struktur der Repositories angepasst werden. Weiterhin ist er für die Projektverwaltungssoftware zuständig und Ansprechpartner beim Bedarf an neuen Funktionalitäten. Er kümmert sich um die Verwaltung von Rollen bzw. Kompetenzen in der jeweiligen Software und ist somit Ansprechpartner, falls weitere Kompetenzen benötigt werden. Datenbanken werden ebenfalls von ihm organisiert. Ferner ist er für die Buildumgebung zuständig und sollte diese instand halten. Falls notwendig, ist er für regelmäßige Backups zuständig.

Schnittstellen

Der Infrastrukturbeauftragte wird z.B. von der Projektleitung über notwendige Funktionalitäten informiert und stellt diese bereit, sofern dies möglich ist. Es herrscht also ein enger Austausch mit dem Projektmanagement. Ferner besteht eine Schnittstelle zum Dokumentenmanagement, da die Infrastruktur für die Dokumente bereitgestellt werden muss. Durch das Erstellen einer gemeinsamen Datenbasis übernimmt der Infrastrukturbeauftragter eine Querschnittsfunktion über alle Bereiche.

2.1.4 Anforderungsmanager

SS 2014 Rolleninhaber: Tobias Kölker und Dag Ennenga, Stellvertreter: Sönke Martens
WS 2014/'15 Rolleninhaber: Kevin Möhlmann, Stellvertreter: Tobias Kölker

Der Anforderungsmanager hat das Ziel ein gemeinsames Verständnis über das zu entwickelnde System zwischen Auftraggeber und -nehmer zu formulieren und aufrecht zu erhalten. Um dieses Ziel zu erreichen, können unterschiedliche Anforderungsmanagementmethoden angewandt werden. Diese Methoden sollen zur Qualität der Dokumentation von Anforderungen beitragen. Das Anforderungsmanagement hilft bei arbeitsteiligen Projekten und dient der effizienten und fehlerarmen Entwicklung.

Zur Verdeutlichung der Anforderungen werden im Allgemeinen Modellierungssprachen (UML, MSC, ...) genutzt und entstandene Modelle im Anforderungsdokument eingefügt. Mit Hilfe von Anforderungen werden Prozesse definiert und implementiert. Während der Implementierung wird das Anforderungsdokument weiter aktualisiert und soll später als Grundlage für Testfälle dienen. Wichtige Qualitätsmerkmale während der Erstellung der Anforderungen sind Verständlichkeit, Eindeutigkeit, Widerspruchsfreiheit, Nachweisbarkeit, Vollständigkeit und Testbarkeit.

Des Weiteren hat der Anforderungsmanager die abschließende Aufgabe, die Erfüllung der einzelnen Anforderungen festzustellen und zu dokumentieren. Damit soll eine Aussage darüber ermöglicht werden, in wie weit das vom Auftraggeber gewünschte System realisiert wurde. Für nicht vollständig umgesetzte Anforderungen sollen die Gründe für das nicht Erfüllen ausgemacht werden.

Schnittstellen

Der Anforderungsmanager dient als Ansprechpartner für alle anderen Projektgruppenmitglieder, denn er kennt die gewünschten Funktionalitäten des zu entwickelnden Systems. Er arbeitet eng mit der Projektleitung zusammen und leitet aus noch nicht umgesetzten Anforderungen Arbeitspakete ab. Um die Erfüllung der einzelnen Anforderungen festzustellen und Probleme zu dokumentieren, tritt er nochmals mit allen PG-Mitglieder in Kontakt.

2.1.5 Softwarearchitekt

SS 2014 Rolleninhaber: Hendrik Grunau, Stellvertreter: Dag Ennenga
WS SS 2014/'15 Rolleninhaber: Sönke Martens, Stellvertreter: Connor Fibich

Der Softwarearchitekt ist für die zu erstellende Softwarestruktur zuständig. Dabei ist seine primäre Aufgabe eine Schnittstelle zwischen Projektgruppe und Softwarearchitektur zu bieten, um für Fragen bei der Integration neuer Module, Aufgabe einzelner Softwareteile oder ähnlichem als Anlaufstelle zu dienen und die Übersicht zu wahren. Insbesondere sei hierbei die Vermittlung zwischen Anforderungsmanagement, Projektmanagement, Dokumentenbeauftragten und Hardwarearchitekten genannt, um angrenzenden Bereichen ausreichend Informationen über den Status der Software zu bieten. Ein weiterer, wichtiger Aufgabenbereich ist das Erstellen der nötigen UML-Diagramme, damit der Projektgruppe die Übersicht über die Software garantiert ist. Abschließende Aufgabe ist darüber hinaus dafür zu sorgen, dass die Mitgliedern der Projektgruppe eine einheitliche Sicht auf die zu erstellende Softwarearchitektur erhalten und implementieren können.

2.1.6 Projektmanager

SS 2014 Rolleninhaber: Claas Martin & Marius Hacker
WS 2014/'15 Rolleninhaber: Christian Best & Michael Cordes

Der Projektmanager ist für die Planung und Steuerung des Projektes zuständig. Zur Planung gehört die Strukturierung der Gruppe, die Urlaubsplanung sowie das Planen der Projekt-Phasen und Iterationen. Nachdem die Planung einer Projekt-Phase abgeschlossen ist und die Realisierung beginnt, kümmert sich der Projektmanager um die Kontrolle der einzelnen Arbeitspakete. Werden Abweichungen zur Planung festgestellt, zum Beispiel wenn der Zeitaufwand falsch geschätzt wurde, müssen Steuerungsmaßnahmen ergriffen werden. Neben der prädiktiven Planung ist dementsprechend auch reaktives Gegensteuern gefragt. Darüber hinaus berichtet das Projektmanagement regelmäßig über den aktuellen Projektfortschritt.

Neben den fachlichen Aspekten kümmert sich der Projektmanager auch um das Projektteam und motiviert es. Während das Projekt gemeinsam durchgeführt wird, fungiert er als Ansprechpartner für alle Mitglieder des Teams. Dazu gehören auch interne Probleme, also die Konfliktbewältigung zwischen Teammitgliedern. Des Weiteren vertritt er das Projekt nach außen.

Schnittstellen

Der Projektmanager arbeitet eng mit allen Disziplinen zusammen. Insbesondere arbeitet er mit dem Anforderungsmanager zusammen, um Anforderungen als Grundlage für die Planung zu nutzen. Ebenfalls besteht eine enge Zusammenarbeit mit dem Dokumentenbeauftragten, um z.B. anstehende Dokumentationsaufgaben in die Planung mit einfließen zu lassen.

2.1.7 Entwicklungsumgebungs bzw. Prozessmanager

SS 2014 Rolleninhaber: Eugen Langolf, Stellvertreter: Christian Best

Der Entwicklungsumgebungsmanager sorgt für die Entwicklungsumgebung. Das umfasst die Einrichtung mittels Anleitung oder vorinstallierter VM und die Verbesserung der verwendeten Tools mittels Plugins oder anderen Erweiterungen. Die Ermittlung der Toolchain zur Verwendung der Hardware ist in der Zusammenarbeit mit dem Hardwarearchitekt zu erledigen. Ebenso spielen die Vorstellungen des Softwarearchitekten bei der Auswahl der Umgebung eine wichtige Rolle. Im Wintersemester wurde die Rolle nicht mehr ausgefüllt. Wartungsarbeiten wurden von dem vorherigen Rolleninhaber bzw. Vertreter ausgeführt.

2.1.8 Hardwarearchitekt

SS 2014 Rolleninhaber: Kevin Möhlmann & Connor Fibich
WS 2014/'15 Rolleninhaber: Connor Fibich, Stellvertreter: Sönke Martens

Der Hardwarearchitekt ist für die Einarbeitung in die Hardwarekomponenten zuständig. Er kennt die Geräte und ihre Funktionen, weiß wie sie verbunden sind und wie sie angesprochen werden. Zudem bietet er eine Schnittstelle zwischen Projektgruppe und der Hardware. Er gibt eine erste Übersicht über die Hardwarekomponenten und hilft bei Anlaufschwierigkeiten. Auch kennt der Hardwarearchitekt die speziellen Anforderungen an die Benutzung der Hardware. Er kennt die zu installierenden Softwarekomponenten, die zum Ansprechen und Programmieren der Hardware dienen. Er weiß, wo sie zu bekommen und wie sie zu installieren sind. Zudem hat er einen groben Überblick über die Benutzung dieser Softwarekomponenten.

2.1.9 Kassenwart

SS 2014 & WS 2014/'15 Rolleninhaber: Tobias Koelker, Kassenprüfer: Christian Best

Der Kassenwart übernimmt die Verantwortung für die Kaffeekasse. Er führt eine Liste über Einnahmen und Ausgaben. Dabei ist diese Liste nicht dem Namen nach auf Kaffee beschränkt, sondern enthält auch Materialienbesorgungen.

2.2 Vorgehensmodell

Viele Projekte scheitern, wie in Abbildung 2.1 dargestellt. Bereits seit einigen Jahren liegt der Anteil der erfolgreich beendeten Projekte bei ca. einem Drittel. Der restliche Teil der Projekte wird entweder nicht zu Ende geführt oder es werden nicht alle Ziele erreicht. Häufig scheitern Projekte an mangelnder Kommunikation oder unklaren Zielvorgaben [10]. Um unser Projekt erfolgreich abzuschließen, haben wir uns dazu entschlossen, mit Projektmanagementmaßnahmen unser Vorgehen zu planen.

Als mögliche Vorgehensmodelle wurden das V-Modell-XT, SCRUM und Unified Process betrachtet. Das von der Bundesregierung geförderte V-Modell-XT nutzen wir nicht, da die benötigten Werkzeuge nicht fehlerfrei liefen. Zum Beispiel funktionierte der Im- und Export von Dokumenten nicht zuverlässig. SCRUM hat sehr große Vorteile bei Projekten, bei denen die Anforderungen am Anfang des Projektes nicht ganz klar sind, bzw. im Verlauf des Projektes noch weitere Anforderungen hinzukommen. Dies ist bei unserem Projekt nicht der Fall. Anforderungen werden sich in unserem Projekt nicht grundlegend verändern. Des Weiteren erschien uns der organisatorische Aufwand, um SCRUM durchzuführen, verhältnismäßig groß.

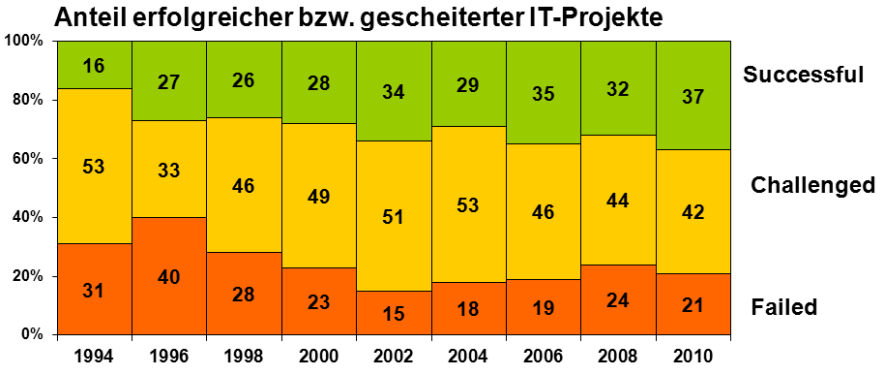


Abb. 2.1 Anteil erfolgreicher bzw. gescheiterter IT-Projekte [64]

Dies kommt, da SCRUM tägliche Treffen und kurze Sprints vorsieht. Dies ist im Alltag der Universität kaum zu erreichen, da die Teammitglieder alle unterschiedliche Stundenpläne haben. Eine vorgesehene Sprintdauer von zwei bis vier Wochen schien uns vor diesem Hintergrunde ebenfalls etwas zu kurz angesetzt zu sein. SCRUM hätte deshalb noch sehr stark auf unser Projekt zugeschnitten werden müssen. Diese Anpassungen schienen uns jedoch so groß, dass wir sie nicht ausprobieren wollten, da ein scheitern der Anpassung auch möglich gewesen wäre. Deshalb haben wir uns für *Unified Process* als Vorgehensmodell entschieden haben. Ein weiterer Vorteil von Unidied Process ist weiterhin, dass Business Modelling und Anforderungsanalyse bereits in das Modell integriert sind. Bei SCRUM müssen diese Punkte eigenständig erarbeitet und in das Rahmenwerk eingebunden werden. Bei Unified Process sind sie feste Bestandteile des Vorgehens und können somit auch nicht so stark in den Hintergrund treten. Dieser Punkt war ein sehr ausschlaggebender, weil Aufgrund der Unerfahrenheit der Teilnehmer des Projektes vermutet wurde, dass die Punkte während der Entwicklung nicht im Fokus stehen. Durch die Wahl von Unified Process haben wir erreicht, dass die Software stets an den Anforderungen angelehnt entwickelt wird.

Unified Process ist ebenso wie SCRUM ein iteratives und inkrementelles Vorgehensmodell. Am Ende jeder Iteration steht ein ausführbares Produkt, das eine Vorversion des Endprodukts ist. Um dieses Produkt zu erstellen, werden in jeder Iteration die folgenden neun Arbeitsschritte ausgeführt.

1. **Geschäftsprozessmodellierung:** Dieser Bereich ist dafür vorgesehen, das Ziel zu verstehen.
2. **Anforderungsanalyse:** Alle Anforderungen, die an das Projekt gestellt werden, werden hier bearbeitet. Zunächst mit dem Auftraggeber erfasst und dokumentiert. Während des Projektes wird kontrolliert, ob die Anforderungen noch korrekt sind und ob sie schon erfüllt sind.
3. **Analyse & Design:** Hier wird der Entwurf der Architektur der Software erstellt.

4. **Implementierung:** Nach einem iterativen Vorgehen wird in diesem Abschnitt die Software entwickelt.
5. **Test:** Die Software wird regelmäßig auf Fehler überprüft.
6. **Auslieferung:** Dieser Abschnitt beinhaltet alle Arbeiten, die nötig sind, um das Produkt an den Kunden ausliefern zu können.
7. **Konfigurations- und Änderungsmanagement:** Änderungen, die während des Projektes auftreten, werden hier verwaltet und in das Projekt eingebracht.
8. **Projektmanagement:** Das Projektmanagement wird während des gesamten Projektes durch den Projektleiter durchgeführt.
9. **Infrastruktur:** Die notwendige Infrastruktur für das Projekt wird installiert und gewartet.

Diese Arbeitsschritte werden in verschiedenen Iterationen verschieden intensiv durchgeführt. Dazu werden einzelne Iterationen Phasen zugeordnet:

1. **Inception:** In dieser Phase wird die Idee des Projektes zu konkreten Zielen und Anforderungen ausformuliert. Dafür werden in dieser Phase vor allem die Arbeitsschritte Geschäftsprozessmodellierung und Anforderungsanalyse durchgeführt.
2. **Elaboration:** In dieser Phase wird ein erster Prototyp fertig gestellt, sowie der größte Teil der Anforderungen definiert. Des Weiteren erfolgt eine Planung der nächsten Phase. Während der Elaboration werden vor allem die Arbeitsschritte Analyse & Design und Anforderungsanalyse ausgeführt. Die Implementierung des Produktes beginnt bereits in dieser Phase.
3. **Construction:** Mit Abschluss der Elaborationsphase sind die meisten Anforderungen festgehalten und werden nun in der Construction-Phase umgesetzt. Es entsteht eine lauffähige Version des Produktes, die gleichzeitig getestet und auf das Release vorbereitet wird.
4. **Transition:** In dieser Phase wird das Produkt an den Kunden ausgeliefert und das Projekt abgeschlossen.

In Abbildung 2.2 ist ein beispielhafter Verlauf eines Projektes mit Unified Process dargestellt. In jeder Phase wird jeder Arbeitsschritt ausgeführt. Die erste Phase besteht jedoch zum größten Teil aus Geschäftsprozessmodellierung und Anforderungsanalyse. Erst in der Elaboration-Phase bekommen die Arbeitsbereiche Analyse & Design und Implementierung mehr Gewicht. Die Bereiche Geschäftsprozessmodellierung und Anforderungsanalyse nehmen am Ende dieser Phase weniger Zeit in Anspruch und laufen bis zum Abschluss des Projektes nur noch nebenher. Der Arbeitsschritt Test wird während des gesamten Projekts durchgeführt. Die Intensität der Durchführung schwankt allerdings. Erst am Ende des Projekts nimmt das Thema Auslieferung an Bedeutung zu. Eine gewisse Bedeutung hat die Auslieferung allerdings während des gesamten Projekts, weil am Ende jeder Iteration dem Kunden ein lauffähiges Produkt präsentiert wird [31, Seite 8].

Diese Vorstellung ist wichtig, damit auf sich ändernde Anforderungen oder Fehlschläge rechtzeitig reagiert werden kann. Am Ende einer Iteration können sowohl wir als auch der Kunde das Produkt begutachten. Falls Probleme an einzelnen Komponenten oder Fehlentwicklungen festgestellt werden, können diese besprochen und

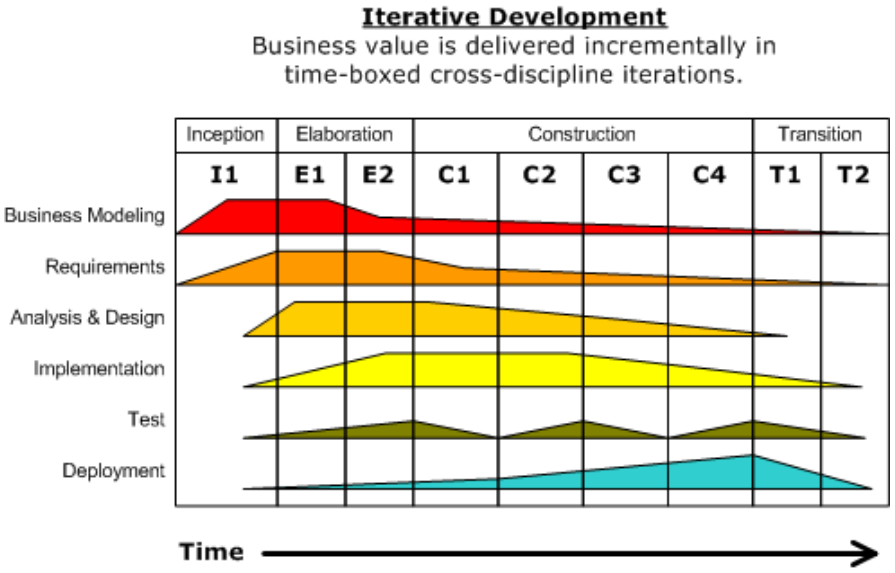


Abb. 2.2 Visualisierung der Intensität der Arbeitsschritte in Iterationen und Phasen [7]

zeitnah korrigiert werden. Ein weiterer Vorteil für uns ist, dass wir durch das stetige Anforderungsmanagement die an uns gestellten Anforderungen stets vor Augen haben.

Im nächsten Abschnitt wird erläutert, wie wir Unified Process in unserem Projekt angewendet haben. Dabei sind kleine Anpassungen notwendig gewesen, weil das Projekt im Rahmen einer Lehrveranstaltung und nicht in einem Unternehmen statt fand.

2.3 Projektablauf

Wir sind nach dem Vorgehensmodell *Unified Process* vorgegangen, wie in Abschnitt 2.2 beschrieben. Zu Beginn des Projekts war der Entwurf eines Projektplan vorgesehen. Dieser sollte bei Bedarf an den tatsächlichen Verlauf angepasst werden. Vor der initialen Projektphase fand eine Seminarphase statt, in der ein Seminarband mit einzelnen Seminararbeiten erstellt wurde.

Ursprünglicher Projektplan

Der Projektplan zum Stand des 17. Oktobers ist in Abbildung 2.3 dargestellt. Dieser musste gegenüber dem ursprünglichen Plan angepasst werden. Aufgrund unserer ge-

ringen Praxiserfahrungen in der Projektplanung wurde die Dauer der Entwurfsphase (engl. Elaboration) im initialen Projektplan unterschätzt. Sie wurde um einen Monat bis Mitte Oktober verlängert.

Im *Unified Process* ist, wie in Kapitel Abschnitt 2.2 vorgestellt, die Modellierung von Geschäftsprozessen vorgesehen. Dies dient dem Verständnis der Organisation, in der das System letztendlich verwendet werden soll. Grundlegende Überlegungen dazu sind in die Anforderungen mit eingangen (s. 4). Eine gesonderte Betrachtung ist im Rahmen dieses Projekts allerdings nicht möglich gewesen, weil keine konkrete Zielorganisation bekannt gewesen ist. [65]

Zu Beginn des Unified Process sind nach Abbildung 2.2 die Phasen *Requirements* und *Analysis & Design* im Fokus. Während also zunächst die Erhebung von Anforderungen wesentliches Ziel war, folgte anschließend bzw. teilweise überlappend ein Prototyp, um bereits erste Anforderungen umzusetzen. Im Projektplan wurde dies durch die Phasen Konzeptualisierung und Entwurf abgebildet. In der Konzeptualisierungsphase wurden zunächst Anwendungsfälle entwickelt. Darauf aufbauend wurde in der Entwurfsphase eine Vision entwickelt und Anforderungen konkretisiert. Gleichzeitig wurde wie vorgesehen bereits mit der Implementierung eines Prototyps begonnen.

In der nächsten Phase des Unified Process-Vorgehensmodells stand die Implementierung im Vordergrund. Dies wurde bei uns durch die von Mitte Oktober bis Ende Januar geplante Implementierungsphase abgebildet, in der eine Alpha-Version, eine Beta-Version sowie die fertige Version entwickelt werden sollten.

Parallel zu den anderen Phasen sollen nach *Unified Process* Tests stattfinden. Daher wurde ebenfalls vorgesehen, bereits während der Implementierung zu testen und bis Ende Januar Testfälle zu entwickeln. Das Vorgehensmodell schließt mit einer *Deployment*-Phase ab, in der das fertige Produkt ausgeliefert werden soll. Dies wird im Projektplan durch die Übergabephase abgebildet. Dort sollte die Dokumentation aufbereitet werden, Bugs behoben sowie die Evaluation abgeschlossen werden.

Somit wurde im ersten Semester der Schwerpunkt auf die Analyse von Anforderungen, der Problemanalyse und das Design gelegt, während im zweiten Semester der Fokus auf der Implementierung lag.

Realisierter Projektplan

In Abbildung 2.4 ist der tatsächliche Projektplan zu Ende des Projekts abgebildet, der sich stellenweise vom Projektplan zum Stand des 17. Oktobers (vgl. Abbildung 2.3) unterscheidet. Die Unterschiede sind im Wesentlichen in Details zu finden. So konnte z.B. keine Beta-Version realisiert werden, während die Phasen insgesamt nur wenig verschoben werden mussten. Details zu den Phasen und deren Anpassung werden im Folgenden erläutert.

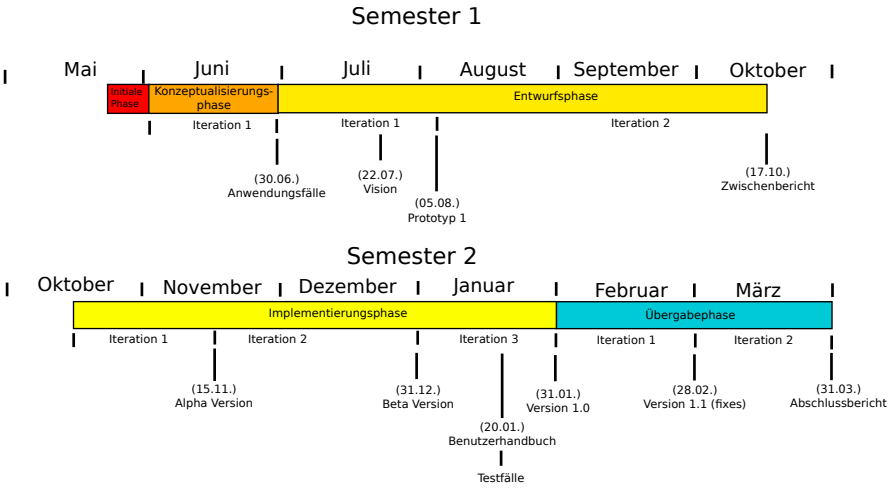


Abb. 2.3 Projektplan - ursprüngliche Planung

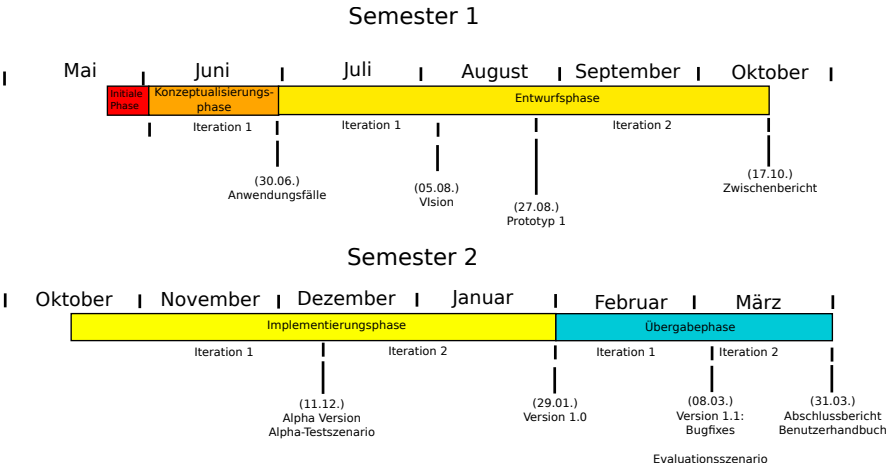


Abb. 2.4 Projektplan - tatsächliche Realisierung

2.3.1 Initiale Projektphase

In der initialen Phase des Projekts vom 27.05. bis zum 03.06.2014 wurden die Rahmenbedingungen für das Projekt festgelegt. Hierzu gehörten Rolleneinteilungen, die Reihenfolge der Moderatoren und Protokollanten, die Festlegung des Vorgehensmodells und die Auswahl eines Ticketsystems.

2.3.2 Konzeptualisierungsphase

Nach der initialen Phase folgt laut Abschnitt 2.2 die Konzeptualisierungsphase (engl. Inception). In dieser Phase wurden vor allem Use-Cases erhoben und die Vision erstellt. Die Vision ist im *Unified Process* ein wichtiges Dokument, da sie das Projektziel beschreibt. Durch das frühzeitige Festlegen der Vision wurde sichergestellt, dass alle Projektbeteiligten wussten, was das konkrete Ziel des Projekts ist.

2.3.2.1 Iteration 1: „Vision, Use-Cases und Anforderungserhebung"

In der ersten Iteration vom 02.06. bis 27.06.2014 wurden Arbeitsgruppen gebildet. Durch die Einteilung in Arbeitsgruppen konnte sich jedes Projektmitglied in einen Aufgabenbereich einarbeiten. Im Folgenden werden die gebildeten Gruppen beschrieben.

Die Gruppe Anforderungsmanagement erhob die Use-Cases sowie Anforderungen und erstellte die Vision. Die Projektmanagement-Gruppe hatte die Aufgabe, die nächste Iteration zu planen und einen Projektplan zu erstellen. Die beiden Gruppen zur fachlichen Einarbeitung in Hard- und Software haben sich mit den Hardwarekomponenten der Firma Beckhoff beziehungsweise der Softwarearchitektur beschäftigt.

Für die Dokumentenvorlagen und die Bereitstellung der Infrastruktur war die Gruppe Dokumentenvorlagen beziehungsweise Infrastruktur zuständig. Sie haben das Ticketsystem *Jira* von Atlassian installiert und eingerichtet, Dokumentenvorlagen erstellt, eine virtuelle Maschine für die Entwicklung eingerichtet und sich um die restliche Infrastruktur für unser Projekt gekümmert.

Ein Ergebnis dieser Iteration sind die ausgearbeiteten Anforderungen in Kapitel 4.

2.3.3 Entwurfsphase

In der Entwurfsphase von Anfang Juli bis Mitte Oktober wurde ein erster Prototyp programmiert, die Anforderungen verfeinert und der Zwischenbericht geschrieben.

2.3.3.1 Iteration 1 - „Prototyp 1“

In der ersten Iteration der zweiten Phase vom 27.06. bis zum 05.08.2014 fand die Einarbeitung in die Programmierumgebung und das Erstellen eines ersten Prototypen statt. Die Entwicklung dieses ersten Prototypen sollte allen Projektmitgliedern die Möglichkeit geben, sich in die Entwicklungsumgebung und die gewählte Programmiersprache einzuarbeiten. Da das verwendeten Vorgehensmodell *Unified Process* einen iterativen Ansatz umsetzt, ist bereits frühzeitig die Implementierung eines Prototypen geplant. Details zu den Zielen des Prototypen sind in Abschnitt 7.1

dargestellt. Als Programmiersprache haben wir uns für C++ entschieden, weil die Simulinkmodelle automatisch in C++ Quelltext umgewandelt werden können und die Mehrheit der Projektmitglieder sich gerne näher mit C++ beschäftigen wollte. Neben der Standardbibliothek haben wir auch die Boostbibliothek eingebunden, die im aktuellen C++ Standard aufgenommen wurden.

Jedem Projektmitglied wurde eine Programmieraufgabe zum Prototypen zugewiesen. Dementsprechend wurde die Entscheidung getroffen, früh mit ersten Entwicklungsschritten anzufangen. Diese Entscheidung hatte das Ziel, frühzeitig Fehler in der Softwarearchitektur und Probleme mit den benutzten Technologien zu erkennen.

Anhand des ersten Prototyps wurden erste Prognosen über den Umfang des gesamten Systems möglich. Es zeigte sich, dass die Implementierung sehr umfangreich werden würde, da verschiedene Systeme in verschiedenen Programmiersprachen entwickelt werden mussten, die zudem miteinander kommunizieren.

Neben dem Implementieren des Prototypen wurde auch weiter in den Arbeitsgruppen gearbeitet. Es wurden z.B. die Anforderungen verfeinert und das Dokument für den Zwischenbericht angelegt.

2.3.3.2 Iteration 2 - „Prototyp 2“

In der zweiten Iteration der zweiten Phase vom 06.08. bis zum 01.10.2014 wurde mit der Entwicklung eines zweiten Prototypen begonnen. Daneben wurde intensiv am Zwischenbericht gearbeitet, weil dieser am Ende der Iteration fertig sein sollte. Als abzusehen war, dass der zweite Prototyp zum Iterationsende nicht vollständig implementiert sein würde, mussten Maßnahmen getroffen werden. Es wurde entschieden, die Konzentration auf den Zwischenbericht zu legen und den Prototypen nicht mehr abzuschließen. Statt der weiteren Entwicklung eines zweiten Prototypen wird in der Implementierungsphase (engl. Construction) direkt mit der Entwicklung einer Alpha Version begonnen, weil die Implementierung des Prototypen ansonsten zu Verzögerungen im Gesamtprojektverlauf geführt hätte.

2.3.4 Implementierungsphase

In der Implementierungsphase, die vom 17.10. bis zum 29.01.2014 dauerte, wurde ein fertiges Endprodukt, die Version 1.0, zu entwickelt. Dazu sollte zunächst eine Alpha-Version erstellt werden, die durch eine Beta-Version erweitert werden sollte. Abschließend sollte die finale Version 1.0 zum 31. Januar fertiggestellt werden.

Wesentlich in der Implementierungsphase war zudem das wöchentliche Stand Up in der internen Sitzung, in dem jeder Projektteilnehmer über seinen Fortschritt berichtete. Ferner wurde der Projektfortschritt zweiwöchentlich mittels einer Excel-Tabelle prozentual aufbereitet und den Betreuern vorgestellt.

2.3.4.1 Iteration 1 - „Alpha Version“

Die erste Iteration der Implementierungsphase, die am 17.10. begann, sollte am 15.11. mit einer Alpha-Version abschließen. Diese Alpha-Version sollte bereits wesentliche Funktionalitäten der fertigen Version beinhalten, wie in Abschnitt 7.2 beschrieben.

Obwohl alle Bestandteile der Systeme einzeln erfolgreich getestet wurden, ergaben sich diverse Probleme bei der Integration der verschiedenen Komponenten, sodass z.B. notwendige Datenbankschnittstellen fehlten oder notwendige Parameter nicht gesetzt wurden. Daher war der Integrationsaufwand auch aufgrund der Komplexität der Systeme deutlich größer als gedacht. Als Gegenmaßnahme wurden mehrere Programmier-Samstage eingeführt, um die Implementierung abzuschließen. Die fertige Alpha-Version konnte schließlich am 11.12. präsentiert werden (vgl. Abbildung 2.4).

Der ursprüngliche Projektplan sah vor, dass zum 31.12. darauf aufbauend bereits eine deutlich erweiterte Beta-Version fertig sein sollte, die zum Beispiel alle Komponentenmodelle enthalten sollte (siehe Abbildung 2.3). Dies war aufgrund der Verzögerungen nicht mehr realisierbar. Daher war eine Umplanung notwendig, sodass direkt bis zum 31. Januar die Fertigstellung der Version 1.0 geplant wurde. Diese Umplanung hatte erneut zum Ziel, Verzögerungen im Gesamtprojektverlauf zu minimieren.

2.3.4.2 Iteration 2 - „Version 1.0“

Im Anschluss an die Alpha-Version wurde dementsprechend direkt mit der Implementierung der Version 1.0 begonnen, die z.B. sämtliche Komponentenmodelle beinhalten sollte (für einen detaillierten Überblick über den Funktionsumfang siehe Abschnitt 7.3). Da keine Beta-Version mehr vorgesehen war, musste mehr Zeit für Bugfixes eingeplant werden. Daher wurde das Ende der Iteration für die Version 1.0 vom 31. Januar auf den 16. Januar vorverlegt. Aufgrund der Prüfungsphase und Ausfall durch Krankheit kam es jedoch zu Verzögerungen, sodass die Version 1.0 letztlich am 29. Januar präsentiert werden konnte. Trotz der Verzögerung lagen wir jedoch wieder - durch das Auslassen der Beta-Version - im Zeitplan des ursprünglichen Projektplans (vgl. Abbildung 2.3). Daher erwies sich die Entscheidung, keine Beta-Version zu entwickeln, insgesamt als richtig.

2.3.5 Übergabephase

Die Übergabephase sieht neben dem Beheben von Bugs die Evaluation des Systems sowie die Dokumentation, d.h. die Erstellung des Abschlussberichtes und des Benutzerhandbuchs, vor.

2.3.5.1 Iteration 1 - „Version 1.1“

Zu Beginn der Übergabephase war es zunächst wichtig, bekannte Bugs zu beheben, da diese z.B. die Ergebnisse der Evaluation beeinflussen könnten. Weiterhin musste für die Evaluation noch eine grafische Benutzeroberfläche (GUI) entwickelt werden, da diese aus der Version 1.0 aus Zeitgründen ausgegliedert wurde. Diese GUI sollte die Betrachtung und den Export, z.B. von Komponentendaten, ermöglichen. Weiterhin sollten Diagramme erstellt und exportiert werden können. Zeitgleich wurde ein Konzept für die Evaluation entwickelt und bereits mit der Dokumentation begonnen.

Für die Umsetzung der Evaluations-GUI waren aufgrund des nahenden Projektendes lediglich zwei Wochen vorgesehen. Um dieses sportliche Ziel zu erreichen wurden 5 Personen mit der Erstellung der GUI beauftragt. Dennoch wurde das Ziel um eine Woche verfehlt, sodass die Iteration schließlich am 23. Februar abgeschlossen werden konnte.

2.3.5.2 Iteration 2 - „Evaluation“

Ab dem 24. Januar sollte die Evaluation umgesetzt werden. Dazu diente das Grobkonzept, welches in der Iteration zuvor erarbeitet wurde. Die Evaluation wurde in drei Testseznarios gegliedert. Während das erste Testszenario Basisparameter evaluiert und daher Voraussetzung für die weitere Evaluation war, konnten die anderen beiden Testzsenarien parallel erarbeitet werden.

Während der Evaluation, insbesondere während des Basisszenarios, traten weitere Bugs, z.B. durch unterschiedliche Einheiten in verschiedenen Komponenten oder sehr langsame Datenbankfunktionen, auf. Diese mussten Anfang März behoben werden, weshalb das Erscheinen der Version 1.1 Abbildung 2.4 auf den 8.3. verschoben wurde.

2.3.5.3 Iteration 3 - „Dokumentation“

Bereits parallel zur Evaluation wurde an dem Benutzerhandbuch, dem Abschlussberichts und der Dokumentation der Evaluation gearbeitet. Am Ende der Iteration war zudem die Planung der Abschlusspräsentation und die Vorbereitung eines Videos erforderlich.

2.4 Infrastruktur

Für den Aufbau der für das Projekt notwendigen Infrastruktur war zunächst die primäre Aufgabe, eine geeignete Software zu finden, welche die Verwaltung, Überwachung und Aufgabenverteilung des Projektes unterstützt.

Da es eine vielfältige Auswahl an projektunterstützender Standardsoftware gibt, haben wir den Auswahlprozess an dem Phasenmodell von Gronau (vgl. hierzu [19, S. 101]) ausgerichtet. Der im Modell vorgesehene Zeitraum von 4 Monaten wurde aufgrund des kurzen Projektzeitraums auf wenige Wochen verkürzt.

Die einzelnen Phasen des Modells sind in Abbildung Abbildung 2.5 dargestellt. Während zunächst das Ziel der Auswahl definiert wird, werden anschließend Anforderungen definiert, die erfüllt werden müssen. Daraufhin erfolgt eine Marktübersicht, in der unterschiedliche Lösungen verglichen und auf wesentliche Anforderungen, etwa die kostenfreie Verwendung, überprüft werden. Die darauf folgende Screening Phase sieht eine Verdichtung der gefundenen Anbieter vor. Dies war jedoch in unserem Fall aufgrund der geringen Anzahl an Alternativen nicht notwendig. Daher wurde nach der Evaluation der Anforderungen direkt mit der Endauswahl fortgefahren, die dann per Abstimmung bestätigt wurde. [19, S. 101]

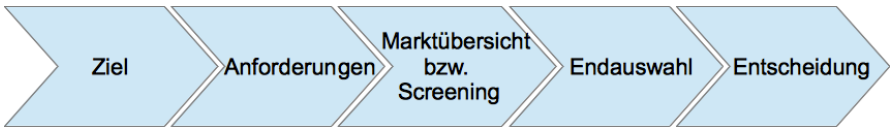


Abb. 2.5 Phasenmodell des Auswahlprozesses (Eigene Darstellung auf Basis von [19, S. 101])

Als Ziel wurde bestimmt, dass die ausgewählte Software das gesamte Vorgehensmodell des Projektmanagements möglichst vollständig abbilden und dessen Umsetzung so effizient wie möglich unterstützen soll (vgl. hierzu Abschnitt 2.2). Dies wurde durch die folgenden, wesentlichen Anforderungen konkretisiert:

1. Buchen und Verwalten von Arbeitszeiten pro Benutzer
2. Einsicht in die Arbeitszeit der Mitglieder durch das Projektmanagement, um z.B. freie Kapazitäten aufzudecken
3. Zuweisen von Aufgaben durch das Projektmanagement, falls notwendig Änderung der Zuweisung durch alle Benutzer
4. Planung von Iterationen
5. Zuordnung von Arbeitspaketen zu Iterationen
6. Unterstützung der Urlaubsplanung
7. Wiki zum gemeinschaftlichen Zusammentragen von Informationen
8. Integration mit der Software zur Versionsverwaltung (z.B. möglichst automatische Zuordnung von Commits zu Arbeitspaketen)
9. Visualisierungstools, die das Projektmanagement vereinfachen (z.B. automatisiertes Erstellen von GANTT-Diagrammen)
10. Einfache und intuitive Bedienbarkeit
11. Erfahrung im Umgang mit der Software durch mindestens ein Projektmitglied, um den Einführungsprozess zu vereinfachen
12. Erweiterbarkeit, z.B. durch Plugins
13. Kostenfreie Nutzung

Bei der anschließenden Marktübersicht kamen für uns aufgrund der Anforderung Nr. 11 lediglich die Anwendungen *Redmine* und *Atlassian JIRA* in Frage, welche in der Lage sind, sämtliche Anforderungen zu erfüllen. Während *Redmine* den Vorteil aufwies, bereits vorinstalliert zu sein, hatte *Atlassian JIRA* Vorteile bei der Bedienbarkeit. Die Abstimmung im Anschluss fiel zugunsten *Atlassian JIRA* aus, welches dann um das Wiki *Confluence* sowie die Versionsverwaltungsunterstützung *Fisheye/-Crucible* sowie um diverse Plugins für das Projektmanagement (z.B. *Agile*, welches optimal mit dem *Unified Process Modell* zusammenarbeitet) erweitert wurde. Als Basis für die Versionsverwaltung haben wir uns nach einem Vortrag über die Funktionalität verschiedener Tools für *GIT* entschieden, da *GIT* z.B. über einen sehr guten Merge Algorithmus verfügt [26].

Um den Einführungsprozess der Software möglichst optimal dem Ziel entsprechend zu gestalten, wurde diese möglichst vollständig an der Projektorganisation ausgerichtet und verschiedene Musterfälle getestet, bevor in den Produktivbetrieb gewechselt wurde.

Um Build-Prozesse zu vereinfachen und qualitativ zu optimieren, wird zusätzlich eine Software zur *Kontinuierlichen Integration* eingesetzt (*Jenkins*). Das Ziel des Einsatzes dieser Software besteht in der Erhöhung der Softwarequalität durch frühzeitiges Aufdecken von Fehlern. Dies geschieht durch eine sofortige Kompilierung des hochgeladenen Source-Codes. [13]

Dies bietet außerdem die Möglichkeit automatisierter Tests, die im Rahmen der Projektgruppe aus Zeitgründen jedoch nicht genutzt wurde. Umgesetzt ist die automatische Kompilierung des Source-Codes und der Dokumentation: Bei Kompilierfehlern werden die Dokumentationsbeauftragten sowie der Verantwortliche für den Fehler per E-Mail informiert.

Damit der Source-Code kompiliert werden kann, wurde ein Jenkins-Node auf einem Windows-PC eingerichtet. Ein Jenkins-Node ermöglicht Jenkins Prozesse auf einem anderen PC auszuführen. Dieser Schritt war nötig, da der uns bereitgestellt Jenkins-Server auf einem Linux-Betriebssystem läuft und unser Projekt unter Windows kompiliert wird.

Die drei Projekte Simulationsframework (SF), Steuerungsebene (SE) und Energiemanagementsystem (EMS) werden unabhängig voneinander kompiliert. Dies ermöglicht eine schnellere Fehlereingrenzung auf eines der Systeme und das Weiterarbeiten an den übrigen, funktionierenden Systemen.

2.5 Qualitätsmanagement

In diesem Kapitel wird das Vorgehen im Rahmen der Projektgruppe für das Qualitätsmanagement erläutert. Dafür wird zunächst in Abschnitt 2.5.1 motiviert, warum ein Qualitätsmanagement überhaupt benötigt wird. In Abschnitt 2.5.2 finden sich Definitionen grundlegender Begriffe, die direkt mit dem Qualitätsmanagement zusammenhängen. Abschließend werden konkrete Maßnahmen des Qualitätsmanage-

ments in Abschnitt 2.5.3 vorgestellt und deren Umsetzung für das einjährige Projekt dokumentiert.

2.5.1 Motivation des Qualitätsmanagement

Dieser Abschnitt beschäftigt sich mit der Frage, warum ein Qualitätsmanagement allgemein für Projekte, in denen Software entwickelt wird, von so großer Bedeutung ist.

Die Fehlerbehebung in der Softwareentwicklung ist in der Regel ein zeitintensiver Faktor, der häufig unterschätzt wird. Die Kosten für die Fehlerbehebung steigen mit zunehmenden Zeitaufwand und können so dazu führen, dass die Realisierung von Projekten behindert wird oder im schlimmsten Fall scheitert. Ein Qualitätsmanagement verfolgt das Ziel, dass auftretende Fehler frühzeitig erkannt werden um Kosten für die Behebung zu minimieren. Qualitätsmanagement als ein Prozess, der ein Projekt in allen Phasen kontinuierlich begleitet, reduziert die zeitlichen Abstände, innerhalb derer Fehler auftreten können und vereinfacht auf diese Art die ansonsten sehr aufwändige Fehlerbehebung. Mögliche Fehlerquellen können alleine mit Hilfe des kleinen Zeitintervalls für die letzten Änderungen auf eine geringe Anzahl reduziert und der eigentliche Fehler schnellstmöglich beseitigt werden. Mehr dazu findet sich in „Eine Untersuchung über Korrekturkosten von Software-Fehlern “[20].

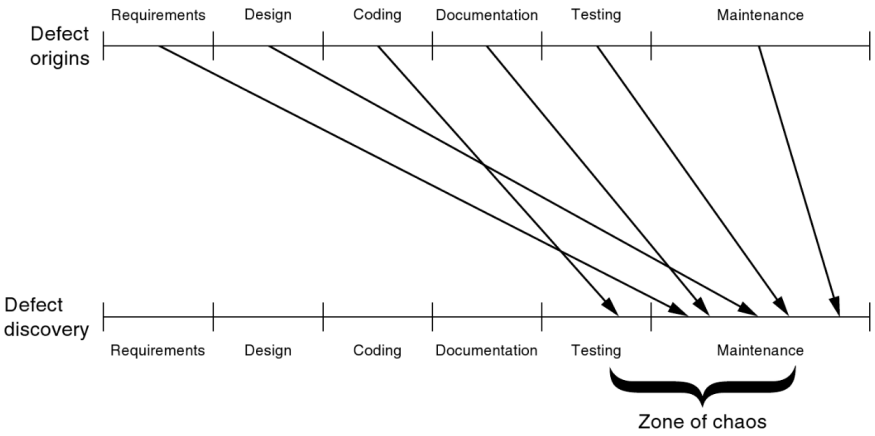


Abb. 2.6 Entstehen und Auftauchen von Fehlern ohne Qualitätsmanagement [30, S. 163]

Die Abbildung 2.6 unterstreicht die Bedeutung des Qualitätsmanagements. Sie verdeutlicht beispielhaft die negativen Auswirkungen auf ein Projekt im Falle einer Vernachlässigung, beziehungsweise Missachtung, der Kriterien für gutes Qualitätsmanagement. Fehler entstehen in einem Projekt ohne Qualitätsmanagement

unbemerkt in allen Phasen der Entwicklung und werden erst in den letzten Phasen realisiert. Die Korrektur dieser Fehler zieht in den meisten Fällen einen sehr großen Zeitverlust nach sich.

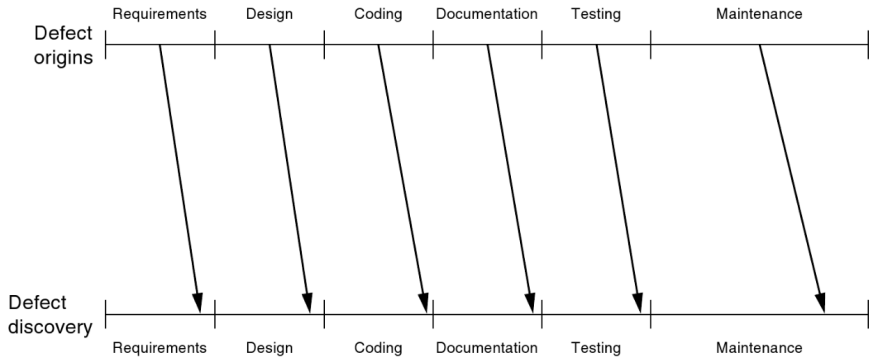


Abb. 2.7 Entstehen und Auftauchen von Fehlern mit Qualitätsmanagement [30, S. 163]

Ziel des Qualitätsmanagement ist es also, wie in Abbildung 2.7 veranschaulicht, Fehler frühzeitig zum Ende jeder der verschiedenen Entwicklungsstufen aufzudecken. Auf diese Art und Weise können diese möglichst schnell behoben werden und das Risiko einer Fehleransammlung reduziert.

2.5.2 Definitionen

In den folgenden Abschnitten werden die grundlegenden Fachausdrücke des Qualitätsmanagements erläutert. Zunächst wird in Abschnitt 2.5.2.1 geklärt, was unter dem Begriff Qualität im Rahmen dieses Berichts verstanden wird. Darauf aufbauend werden in Abschnitt 2.5.2.2 und Abschnitt 2.5.2.3 die Begriffe Qualitätsmanagement und Qualitätssicherung konkretisiert.

2.5.2.1 Qualität

Qualität nach DIN 55350 zielt auf die Erfüllung von Erfordernissen durch ein Produkt oder eine Tätigkeit ab [65, S. 193] und beschreibt, wie sehr sich das Produkt oder eine Tätigkeit dafür eignet. Diese erste Definition erscheint einleuchtend, allerdings verbirgt sich hinter dieser Definition eine große Komplexität. Das bedeutendste Elemente der Definition von Qualität veranschaulicht die Abbildung 2.8 - die Zufriedenheit des Kunden. Laut DIN Norm muss sich das System oder die Dienstleistung

für die Ziele des Kunden eignen. In diesem Zusammenhang muss geklärt werden, wann ein Kunde zufrieden ist beziehungsweise was ein Kunde als Resultat erwartet.

Abb. 2.8 Comic zur Qualität [60]

Verschiedene Nutzer haben verschiedene Erwartungen an Produkte, weshalb Qualität mehrdimensional ist. Verschiedene Faktoren, wie Wartbarkeit, Langlebigkeit oder Effizienz sind für die Qualität relevante Eigenschaften. Qualität wird noch komplexer dadurch, dass diese Faktoren in Abhängigkeit zueinander stehen. Ein besonders genaues System wird einen höheren Zeitbedarf haben. Somit muss hier ein Qualitätsmerkmal gegen ein anderes abgewogen werden und dabei ein akzeptabler Kompromiss gefunden werden [65, S. 141].

Neben sich widersprechenden Qualitätseigenschaften wird Qualität auch komplexer durch verschiedene Nutzer. Deutlich wird dies in der Definition von Qualität von Weinberg nach [65]. Dieser definiert Qualität als die Übereinstimmung der Anforderungen einer Person an ein System mit den tatsächlichen Eigenschaften des Systems. In dieser Definition wird Wert darauf gelegt, dass verschiedene Nutzer unterschiedliche Anforderungen an ein System haben. Während dem einen Nutzer ein besonders genaues Ergebnis wichtig sein kann und er gerne dafür mehrere Minuten wartet, kann einem anderen Nutzer ein schnelles Ergebnis wichtig sein, dass lediglich die Größenordnung der Lösung enthält [65, S. 140 - 141]. Um konfliktäre Anforderungen bei Nutzern aufzudecken, empfiehlt sich besonders ein Anforderungsworkshop, wie in Abschnitt 11.5.3 des Seminarbands beschrieben, in dem verschiedene Nutzer ins Gespräch kommen können und dadurch die Einigung auf Kompromisse erleichtert wird.

2.5.2.2 Qualitätsmanagement

Qualitätsmanagement hat sich seit den 1920er Jahren entwickelt. Zunächst bestand es nur aus der Qualitätskontrolle. In den 1940er Jahren wurde die Qualitätssteuerung hinzugefügt, in den 1960er Jahren die Qualitätssicherung, in den 1980er Jahren das Qualitätsmanagement und als neuste Entwicklung in den 2000er Jahren das Total

Quality Management (TQM). Im TQM wird das Qualitätsmanagement als Managementaufgabe angesehen, die als eine der ersten Aufgaben in Projekten etabliert wird. TQM legt viel Wert auf die Kundenzufriedenheit, die Betrachtung der Supply Chain, auch auf ökologische und kulturelle Aspekte und zielt dabei auf eine ständige Verbesserung ab [51, S. 42 - 44].

Wie in Abbildung 2.9 verdeutlicht, orientiert sich TQM bei der Verbesserung an den Mitarbeitern, den Prozessen, den Kunden und den Ergebnissen, um damit ein möglichst gutes Ergebnis zu erzielen. Dabei sieht das TQM eine Wechselbeziehung zwischen den verschiedenen Orientierungen [51, S. 59 - 63].

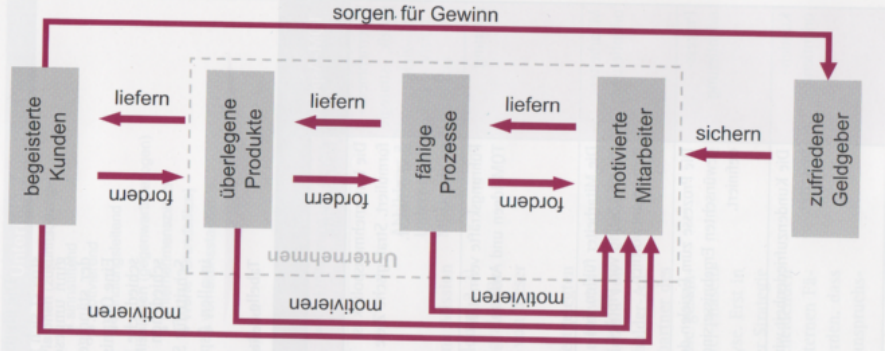


Abb. 2.9 Kausalzusammenhänge im TQM [51, S. 55]

Eine weitere Untergliederung des Qualitätsmanagements und damit ein vertieftes Verständnis des Qualitätsmanagements kann durch den PDCA Zyklus erreicht werden. Der PDCA Zyklus unterteilt das Qualitätsmanagement in verschiedene Phasen und zwar Plan, Do, Check und Act. Im Rahmen einer kontinuierlichen Qualitätsverbesserung, wie beim TQM, werden diese Phasen immer wieder durchlaufen [51, S. 81]. In der Phase Plan wird zunächst die Ist-Situation analysiert und Verbesserungsmaßnahmen beschlossen. Dazu werden Daten zur Feststellung der Ist-Situation erhoben und analysiert. Auf Basis dieser Daten werden Maßnahmen beschlossen. Im nächsten Schritt, dem Do, werden diese Maßnahmen durchgeführt. Mögliche Maßnahmen kann die Qualifikation der Mitarbeiter umfassen. Im Check werden die Ergebnisse dieser Maßnahmen bewertet, indem der neue Ist-Zustand erhoben und analysiert wird. Im Schritt Verbessern wird auf Basis der Daten aus Check entschieden, ob die Maßnahmen die Qualität erhöht haben. Wenn sie die Qualität erhöht haben, so werden sie standardisiert und eingeführt. Ansonsten müssen die Maßnahmen durch weitere Durchläufe des PDCA Zyklus verbessert werden [51, S. 35].

Eine Verfeinerung des PDCA Zyklus ist der DMAIC Zyklus. Die verschiedenen Teilschritte können auf den PDCA Zyklus abgebildet werden und sind inhaltlich sehr ähnlich. Diese Verfeinerung wird in Abbildung 2.10 verdeutlicht. DMAIC steht für Define, Measure, Analyse, Improve und Control. Diese Schritte sind die Grundaktivitäten im Aktivitätsdiagramm. Sie finden sich beim oben beschrieben PDCA

Zyklus ebenso wieder, nur dass das Messen und die Analyse beim PDCA Zyklus mit zur Planning Phase gehören. Der DMAIC Zyklus hingegen fasst Check und Act in die Phase Control zusammen. Mit jeder dieser Phasen sind verschiedenste Methoden verknüpft, die angewendet werden. Im Abschnitt 2.5.3 werden einige Methoden vorgestellt und für das Projekt relevante Methoden identifiziert [51, S. 93 - 95].

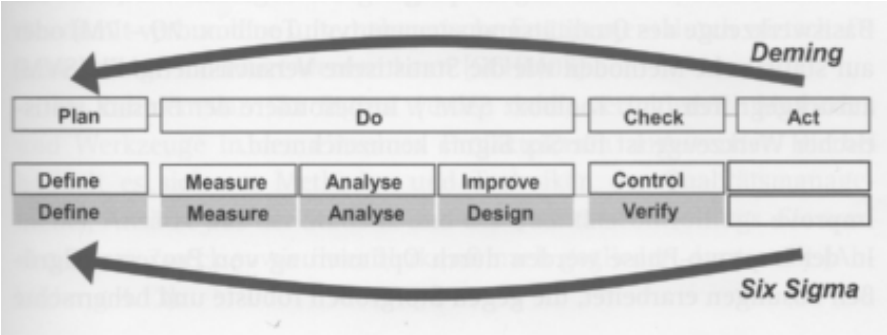


Abb. 2.10 DMAIC Zyklus [51, S. 93]

2.5.2.3 Qualitätssicherung

Um Qualität sicherzustellen, werden verschiedene Maßnahmen ergriffen. Es wird hierbei zwischen konstruktiver Qualitätssicherung und analytischer Qualitätssicherung unterschieden. Unter konstruktiver Qualitätssicherung werden vorbeugende Maßnahmen zur Verhinderung von Fehlern verstanden. Konstruktives Qualitätsmanagement kann auch in Projektmanagement, Prozessmanagement und Testmanagement unterteilt werden [65, S. 143] [38].

Das Testmanagement beschäftigt sich damit, auf welche Art und Weise ein Produkt getestet wird. Näher wir dies in Abschnitt 2.6 erläutert. Das Prozessmanagement setzt bei Prozessen in einem Unternehmen an, um die Qualität sicherzustellen. Unter Prozess werden sowohl die notwendigen Tätigkeiten wie auch die Beschaffung der notwendigen Ressourcen zur Schaffung eines Mehrwerts verstanden. Die Qualität dieses Mehrwerts, wie zum Beispiel ein Produkt, hängt dabei stark von den zugrunde liegenden Prozessen ab. Deshalb wird bei der Erhöhung der Qualität eines Produkts bei den zugrundeliegenden Prozessen angesetzt. Diese werden optimiert, indem die Teilschritte in einem Prozess verbessert werden, indem zum Beispiel Teilschritte aussortiert werden, hinzugefügt werden oder die Reihenfolge geändert wird. Auf diese Art und Weise werden immer besser strukturiere Prozesse geschaffen, die ineinander greifen [23, S.75-95].

Durch ein Vorgehensmodell zum Projektmanagement, wie in Abschnitt 2.2 erläutert, wird die Qualität sichergestellt, indem typische Fehlerquellen vermieden

werden. So wird in Rational Unified Process iterativ vorgegangen, um immer wieder Zwischenergebnisse den Kunden präsentieren zu können und Missverständnisse zu vermeiden. Dadurch wird die Qualität sichergestellt. Deshalb konzentrieren sich die konstruktive Qualitätssicherung auf das Prozessmanagement, wie auch in Abschnitt 2.5.3 erläutert wird [65, S. 143] [38].

Beim analytischen Qualitätsmanagement wird die Qualität eines schon vorhanden Produkts analysiert. Es sind also Maßnahmen, die im nach hinein ergriffen werden, wie die Durchführung von Tests, Reviews oder statischen Codeanalysen. Durch die Maßnahmen soll gemessen werden, wie hoch die Qualität des Produkts ist und mögliche Verbesserungspunkte aufzeigen [38].

2.5.3 Qualitätssicherungsmaßnahmen

Im Folgenden werden im Projekt ergriffene Maßnahmen zur Sicherstellung der Qualität beschrieben. In Abschnitt 2.5.3.1 werden konstruktive bzw. vorbeugende Maßnahmen beschrieben, während in Abschnitt 2.5.3.2 analytische bzw. messende Maßnahmen erläutert werden.

2.5.3.1 Konstruktive Maßnahmen

Ausgehend von im Abschnitt 2.5.2.2 vorgestellten DMAIC Zyklus wird in diesem Abschnitt das Prozessmodell zur Qualitätssicherung von Prozessmodellen konstruiert, das in Abbildung 2.11 zu sehen ist. Der DMAIC Zyklus wird gegenüber dem PDCA Zyklus bevorzugt, da dem PDCA Zyklus das Element des Messens fehlt, der für ein Projekt am Anfang entscheidend ist, weil noch nicht gemessen worden ist. Darüber hinaus lohnt sich das Act bei einem kleinen Projekt wie diesem nicht. Maßnahmen werden meistens gleich für alle umgesetzt und müssen nicht erst standardisiert werden. Aus diesem Grund wird der DMAIC Zyklus verwendet, der genau den Wert auf die Aspekte legt, die für ein kleines und am Anfang stehendes Projekt wie unseres entscheidend sind.

Neben dem DMAIC Zyklus wird auch das Vorgehen des Prozessmanagements nach [23] berücksichtigt. Das hier erarbeitete Prozessmodell soll als vorbeugende Maßnahme auf Prozesse angewendet werden, um in den Prozessen die Qualität sicherzustellen, wie in Abschnitt 2.5.2.3 beschrieben. Das Modell kann und soll auch auf sich selbst angewendet werden.

Im Folgenden wird die Abbildung 2.11 Aktivität für Aktivität beschrieben und erläutert. Dabei wird der Zusammenhang zum DMAIC Zyklus und dem Vorgehen im Prozessmanagement hergestellt.

In der Phase Define sollen die Ziele des Qualitätsmanagements festgelegt werden. Eine dafür geeignete Methode ist die SIPOC-Analyse. SIPOC steht für Supplier, Input, Process, Output und Customer. In dieser Analyse werden alle Teilschritte eines Prozesses, sowie die in den Prozess hineinfließenden und herausfließenden

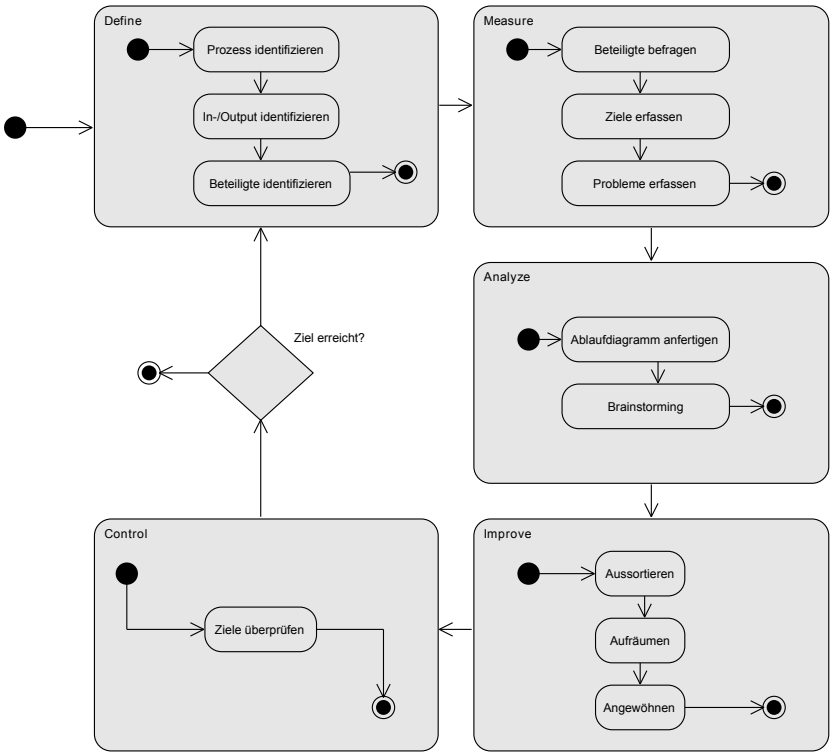


Abb. 2.11 Prozessdiagramm Qualitätsmanagement

Produkte und die an dem Prozess beteiligten Personen festgehalten und beschrieben. Ziel ist die Bildung eines einheitlichen Problemverständnisses und die Eingrenzung der Problemstellung [51, S. 793 - 795].

Die nächste Phase heißt Measure. Hier sollen im klassischen DMAIC Zyklus messbare Größen bestimmt und erfasst werden, um Verbesserungspotenzial durch Vergleich des Ist- und Sollzustand dieser Größen zu ermitteln. Eine Methode dafür ist die CTQ Analyse. Es werden Eigenschaften identifiziert, die für die Qualität des Prozessoutputs kritisch sind [51, S. 791]. Dies findet sich in dem Aktivitätsdiagramm unter der Aktivität Measure in “Beteiligte befragen“ und “Ziele erfassen“ wieder. In diesen Teilschritten sollen kritische Größen erfasst werden. Weitere mögliche Methoden, wie beispielsweise Fehlerlisten, werden ebenfalls durch die Befragung der Beteiligten erledigt, indem nach bisher aufgetretenen Problemen während des Prozesses gefragt wird [51, S. 740]. Diese Befragung dient auch gleichzeitig zur intensiven Kommunikation mit den Teilnehmern des Projekts, da eine Einbeziehung

der Teilnehmer in dem Total Quality Management, wie bereits dargestellt, eine entscheidende Rolle spielt.

Die beiden Phasen Define und Measure finden sich in der ersten Phase des Prozessmanagements wieder, in dem die Prozessarbeit inhaltlich und organisatorisch vorbereitet werden soll [23, S. 96].

Die Ergebnisse der Phase Measure werden für die Phase Analyze genutzt, um ein Verständnis für den Prozess und die kritischen Elemente in dem Prozess zu erwerben. In dieser Phase finden Methoden zur genaueren Beschreibung genauso wie Kreativitätsmethoden Anwendung. [51, S. 94] Von den verschiedenen Methoden eignen sich für unser Projekt insbesondere das Anfertigen eines Ablaufdiagramms. Für dieses werden die am Prozess beteiligten Personen interviewt und dabei die verschiedenen Teilschritte des Prozesses identifiziert und notiert. Diese Methode wird oft durch Methoden zur Prozessoptimierung ergänzt [51, S. 747 - 748]. Die Phase Analyze entspricht der zweiten Phase im Prozessmanagement. In dieser wird der Ist-Zustand der Prozesse beschrieben [23, S. 95].

Die Prozessoptimierung findet in der Phase Improve statt. Im DMAIC Zyklus erfolgt diese Verbesserung durch die Veränderung von Prozessstellgrößen [51, S. 94]. Diese Prozessstellgrößen sind bei uns nicht vorhanden, weshalb wir die Prozesse selbst optimieren, indem wir nicht notwendige Schritte identifizieren und aussortieren und das Prozessmodell neu strukturieren, indem zum Beispiel identifiziert wird, welche Schritte sich parallelisieren lassen. Abgeschlossen wird die Phase Improve durch das Angewöhnen dieses neuen Prozessmodells. Dieser Schritt steht außerhalb des Qualitätsmanagement selbst und muss über eine längere Zeit durch Leben des Prozesses stattfinden. Im Prozessmanagement findet sich die Phase Improve in der dritten Phase wieder. In dieser werden die vorhandenen Prozesse optimiert [23, S. 95].

In der Phase Control wird überprüft, ob die zuvor gesteckten Ziele erreicht werden und ob die identifizierten Fehler noch auftreten oder ob sie behoben wurden. Diese Phase und der iterative Charakter des Modells entsprechen der vierten Phase des Prozessmanagements, in dem eine kontinuierliche Verbesserung vorgesehen ist [23, S. 95].

2.5.3.2 Analytische Maßnahmen

Bei den analytischen Maßnahmen handelt es sich um diagnostische Maßnahmen zur Messung und Sicherung der Qualität des entstehenden Produkts. Sie können auf alle Entwicklungsphasen von der Problemanalyse und den Anforderungen bis hin zur Evaluation und dem Testen der Software bezogen werden. Gängige analysierende Verfahren sind (Produkt-)Reviews, Walkthroughts und Code-Inspektionen. Die ersten beiden Verfahren dienen der formalen Überprüfung schriftlicher Dokumente (zum Beispiel Anforderungen), die im Rahmen des Projekts entstanden sind. Mit Code-Inspektionen wird geschriebener Programm-Code genau analysiert um formale oder inhaltliche Mängel zu identifizieren.

Innerhalb des Projekts werden die drei genannten Verfahren gebraucht, um die Qualität des Produkts dauerhaft sicherzustellen. In den anfänglichen Projektphasen werden die analytischen Maßnahmen Review und Walkthrough in regelmäßigen Abständen auf das Visionsdokument und die Kapitel Problemanalyse und Anforderungen angewendet. In Diskussion zwischen den Gruppenmitgliedern und den Betreuern der Projektgruppe, die auch als Kunden fungieren, entsteht die eindeutige und vollständige Vision, die als Kerndokument der weiteren Arbeit dient. Das Visionsdokument gewährleistet, dass sowohl Auftragnehmer, als auch Auftraggeber ein gemeinsames Verständnis des zu entwickelnden Produkts bekommen. Im Verlauf der Problemanalyse und der damit zusammenhängenden Definition der Anforderung sollen Reviews in verschiedenen Sitzungen und gesonderten Treffen dafür sorgen, dass Qualitätsmerkmale garantiert werden. Dazu gehört die Suche nach Fehlern, Inkonsistenzen, Unvollständigkeiten und weiteren Merkmalen. Im weiteren Verlauf des Projekts finden weiter Treffen statt zur Durchführung der Maßnahmen Review und Walkthrough um die Qualität von Änderungen an bestehender oder neu erstellter Dokumentation zu beurteilen und eventuell auftretende Mängel beheben zu können.

Mit Beginn der Implementierung des Prototypen wird die Code-Inspektion zur wichtigsten analytischen Maßnahme des Projekts. Der Lesbarkeit dienen Code-Guidelines, die von der Projektgruppe bestimmt und eingehalten werden. Die Inspektionen werden einzeln oder in kleinen Gruppen durchgeführt. Sie werden in Abstimmung mit dem Testmanagement (siehe Abschnitt 2.6) durchgeführt.

2.6 Testmanagement

In diesem Abschnitt wird das Vorgehen im Testmanagement beschrieben. Dafür wird in Abschnitt 2.6.1 motiviert, warum und wofür Testmanagement notwendig ist. Danach werden die grundlegenden Begriffe im Umfeld des Testmanagements in Abschnitt 2.6.2 erläutert. Daraus ergibt sich das in Abschnitt 2.6.2.3 erläuterte Prozessmodell. Nachdem der erste Teil gängige Maßnahmen im Testmanagement beschreibt, werden abschließend die während des Projekts gewählte Methoden des Testmanagement in Abschnitt 2.6.3 geschildert.

2.6.1 Motivation

Fehler sind leichter zu beheben, wenn sie früher auffallen, wie in Abschnitt 2.5.1 begründet. Eine Möglichkeit Fehler bzw. Fehlerwirkungen möglichst früh aufzudecken sind Tests. Dadurch können sie erheblich dazu beitragen, die Qualität eines Produkts zu erhöhen, indem sie Fehlerwirkungen im Produktivbetrieb vorbeugen. Zudem ermöglichen es Tests überhaupt, die Qualität von Produkten mit Hilfe verschiedener Metriken zu messen. Durch diese Messung wird das Vertrauen in das Produkt erhöht. [53, S. 9]

2.6.2 Definition

In diesem Abschnitt werden die grundlegenden Begriffe im Umfeld des Testmanagements erläutert. Dies sind in Abschnitt 2.6.2.1 der Fehlerbegriff, in Abschnitt 2.6.2.2 der Testbegriff und in Abschnitt 2.6.2.4 der Begriff Testmanagement. Abschließend werden die verschiedenen für das Testmanagement notwendigen Dokumente erläutert. In Abschnitt 2.6.2.5 wird die Testpolitik erläutert, in Abschnitt 2.6.2.6 das Testhandbuch und in Abschnitt 2.6.2.7 das Testkonzept.

2.6.2.1 Fehlerbegriff

Umgangssprachlich wird von Fehlern häufig in verschiedenen Zusammenhängen gesprochen. Ein Fehler ist in einer Software aufgetreten oder der Entwickler hat einen Fehler gemacht. Für das Testmanagement ist es notwendig präzisere Begriffe zu verwenden, damit genau klar wird, was gemeint ist.

Ein für den Benutzer sichtbarer Fehler wird Fehlerwirkung genannt. Diese Fehlerwirkungen treten beim Testen oder Produktivbetrieb auf und sind auf einen Fehlerzustand in der Software zurückzuführen. In der Software ist möglicherweise eine fehlerhafte Anweisung vorhanden oder ein Ausnahmefall wurde nicht berücksichtigt. Dieser Umstand wird als Fehlerzustand bezeichnet und kann eine Fehlerwirkung zur Folge haben. Ein Fehlerzustand muss aber keine Fehlerwirkung zur Folge haben, weil ein anderer Fehlerzustand den Fehler maskiert kann. Diese Fehlermaskierung bleibt solange nicht sichtbar, bis einer der beiden Fehler behoben wird. [53, S. 7 - 8]

Die Fehlerzustände wiederum sind auf eine Fehlhandlung zurückzuführen. Der Entwickler hat vergessen, sich Gedanken über Ausnahmefälle zu machen oder hat bei der Programmierung eine Kommazahl mit 1,7 eingegeben, anstatt die englische Schreibweise 1.7 zu verwenden. [53, S. 8]

2.6.2.2 Testbegriff

Tests haben die Funktion, gezielt und systematisch Fehlerwirkungen aufzudecken. Somit wird unter Tests im engeren Sinne die Ausführung von einem Testobjekt verstanden, die der Aufdeckung von Fehlerwirkungen dient. Nach Abschluss dieser Ausführung wird das Ist- und Soll Verhalten miteinander verglichen, um eventuelle Abweichungen feststellen zu können. Im weiteren Sinne wird sowohl die Ausführung des Testobjekts zur Bestimmung der Anforderungsüberdeckung oder der Qualität, wie auch die statische Analyse des Testobjekts als Test verstanden. [53, S. 8 - 10]

Teststufen

Tests können auf verschiedene Testobjekte angewendet werden. Eine Einteilung in Teststufen ist über den Umfang des Testobjekts möglich. Wird ein gesamtes System getestet? In diesem Fall ist der Test ein Systemtest. Beim Systemtest wird das System mit allen Bestandteilen getestet. Ziel ist es festzustellen, ob das Verhalten des Systems den spezifizierten Anforderungen entspricht. Dabei sollte das Testobjekt soweit möglich dem späteren Produktivsystem gleichen, um Fehler in dem Zusammenspiel aller Module aufzudecken und auch die Konfiguration mit zu testen. Darüber hinaus ermöglicht dieser Test Last- und Performanztests. Beim Systemtest sollten auch weitere Lieferbestandteile, wie das Benutzerhandbuch und die Dokumentation getestet werden. [53, S. 60 - 63] Dies bedeutet auf unser Projekt bezogen, dass beispielsweise auch das Deployment der ausgelieferten Artefakte getestet werden muss, um sicherzustellen, dass die bereitgestellten Anleitungen so umsetzbar sind.

Bevor das System als Ganzes getestet sind, wird zuvor das Zusammenspiel einzelner Module in einem Integrationstest getestet. Hierbei soll sichergestellt werden, dass die Module so wie geplant zusammen arbeiten können. Es können Fehlerzustände in den Schnittstellen, den ausgetauschten Daten oder auch der Kommunikation vorliegen. Zudem beugen die Integrationstest einem Big Bang beim Systemtest vor. Wenn erst beim Systemtest das Zusammenspiel der Module getestet wird, so können unerwartet viele Fehlerzustände auftreten, es ist schwerer den Fehlerzustand zu lokalisieren und zudem kann die Programmierung der Module lange zurück liegen. Dies hat zu Folge, dass eine Behebung der Fehlerzustände durch den Programmierer zeitaufwändiger ist. [53, S. 52 - 59]

Allerdings muss bei der Durchführung der Integrationstests auch darauf geachtet werden, dass der Testaufwand in einem sinnvollen Verhältnis zu dem erwarteten Nutzen steht. Um dies sicherzustellen, können verschiedene Integrationsstrategien angewendet werden. So werden in der Ad-hoc-Integration die verschiedenen Module sofort nach ihrer Fertigstellung integriert. Der Nachteil bei dieser Strategie ist, dass viele Platzhalter für noch nicht fertiggestellte Module erstellt werden müssen und diese zufällige Reihenfolge in der Regel keiner sinnvollen Reihenfolge entspricht. [53, S. 56 - 59]

Statt dieser zufälligen Reihenfolge ist eine Top-Down oder Bottom-Up Integration möglich. Dies bedeutet, dass die Integration entweder bei dem einzigen nicht aufrufenden Modul, also Bottom-Up oder bei dem einzigen nicht aufgerufenen Modul, also Top-Down gestartet wird. Der Vorteil der Bottom-Up Strategie ist, dass keine Platzhalter für noch nicht fertiggestellte Module erstellt werden müssen. Dafür müssen aber Testtreiber erstellt werden, welche die unterliegenden Module nutzen. Bei der Top-Down Integration sind solche Treiber nicht notwendig, dafür müssen allerdings Platzhalter für noch nicht fertiggestellte Module bereitgestellt werden. [53, S. 56 - 59]

Als weitere mögliche Integrationsstrategie kann die Backbone-Integration gewählt werden. Bei dieser Strategie wird ein Backbone System implementiert, in das die jeweils fertiggestellten Module integriert werden können. Der Vorteil ist, dass die Module direkt nach der Fertigstellung integriert und getestet werden können.

Der Nachteil ist, dass ein solches Backbone System zunächst einmal erstellt werden muss, was durchaus zeitaufwändig sein kann. [53, S. 56 - 59]

In Modultests werden die Bausteine der Entwicklung das erste Mal systematisch getestet. Welche Bausteine dies genau sind, ist abhängig von der Programmiersprache. [53, S. 44 - 51] In C++ ist die kleinste testbare Einheit eine Klasse und damit sind Unittests Modultests. Allerdings ist es üblich lediglich die nach außen sichtbaren Funktionen zu testen. In unserem Projekt werden die nach außen sichtbaren Funktionen einer Releaseeinheit über ein Interface zuvor definiert, wie in Abschnitt 6.1 beschrieben. Deshalb werden in diesem Projekt unter Modultests die Tests gegen diese Interfaces verstanden. Das Verhalten unterhalb dieser Interfaces wird nicht getestet, weil dies wie oben beschrieben nicht allgemein üblich ist und damit Implementierungsdetails getestet werden würden. Diese Implementierungsdetails sind aber Bestandteile der Lösung und nicht der Anforderungen. Somit wären Tests gegen einzelne Klassen nicht nur sehr zeitaufwändig, sondern es gibt auch keinen Sollzustand in Form von Anforderungen, gegen die getestet werden kann. Dies widerspricht auch nicht der Definition nach ISTQB, weil eine Modul sich aus mehreren Bausteinen zusammensetzen kann, solange nur modulinterne Aspekte getestet werden. [53, S. 45]

Bei Modultests soll nur das Modul selbst getestet werden. Dies bietet den Vorteil, dass sich auftretende Fehlerzustände klar einem Modul zuordnen lassen und das Debuggen wird dadurch vereinfacht. Darüber hinaus können diese Tests auch als Regressionstests angewendet werden. Das bedeutet bei Änderungen an einem Modul können deren Tests ausgeführt werden, um festzustellen, ob die neue Implementierung noch den Spezifikationen entspricht. Wichtig sind bei Modultests nicht nur rein funktionale Tests, sondern auch Tests auf Robustheit. Bei diesen Tests wird geprüft, wie die Module auf unerwartete Zustände oder Fehleingaben reagieren. Weiterhin kann getestet werden, wie effizient der Code arbeitet und wie wartbar der Code ist. Dabei sind neben Tests, die den Code ausführen, auch Tests notwendig, die den Code analysieren können. Eine Unterscheidung in statische und dynamische Tests finden sich in den folgenden Abschnitten. Es ist möglich die Testfälle basierend auf dem Code zu schreiben, also so genannte Whitebox Tests zu entwerfen. Allerdings sind diese sehr zeitaufwändig und nach Änderungen in der Implementierung sind diese Testfälle möglicherweise auch nicht mehr sinnvoll. [53, S. 44 - 51]

Statischer Test

Statische Tests sind Tests im weiteren Sinne. Hier wird nicht der Code ausgeführt, wie bei Tests im engeren Sinne, sondern der Code wird analysiert. Diese Analyse erfolgt entweder durch eine Betrachtung von Personen oder durch entsprechende Werkzeuge. [53, S. 81]

Eine strukturierte Analyse durch Personen wird als Review bezeichnet. Diese Personen versuchen den Code oder Dokumente nachzuvollziehen und dabei mögliche Fehlerzustände im Code oder unklare Formulierungen in Dokumenten zu entdecken. Neben der Entdeckung von Fehlerzuständen haben Reviews auch den Vorteil einen

Wissenstransfer zu fördern, weil durch das Review der Personenkreis vergrößert wird, der den Code kennt und warten kann. Auch fördert allein das Wissen darum, dass jemand anderes den Code / die Dokumente lesen wird, dazu das sauberer dokumentiert und formuliert wird, sowohl in Dokumenten, wie auch im Code. Zu beachten ist bei Reviews, dass der Sachverhalt, also das Dokument oder der Code im Vordergrund steht und sachlich darüber gesprochen wird. Ansonsten besteht die Gefahr, dass der Autor unter psychologischen Druck gerät, der hinderlich sowohl für das Review, wie auch die weitere Zusammenarbeit ist. Ein Review muss vorbereitet werden, indem das Ziel des Reviews bestimmt wird und ein passender Personenkreis bestimmt wird. [53, S. 81 - 91]

Ein Review kann als Walktrough gestaltet werden, bei dem der Autor seine Arbeit vorstellt. Möglich ist auch ein formelleres Vorgehen in Form einer Inspektion. Hier ist der Ablauf des Reviews genau durch strukturiert und es wird nach festgelegten Prüfkriterien geprüft. Möglich ist auch ein technisches Review, bei dem überprüft wird, ob das Prüfobjekt mit der Spezifikation übereinstimmt. [53, S. 91 - 98]

Ähnlich dem Review dient auch die statische Analyse der Aufdeckung möglicher Fehlerzustände im Code. Allerdings wird die Analyse nicht durch Personen vorgenommen, sondern durch Werkzeuge, wie zum Beispiel Rechtschreibprogramme oder Compiler. Diese statische Analyse sollte möglichst vor Reviews durchgeführt werden, um leicht zu entdeckende Fehlerzustände schon vorher zu korrigieren. Um überprüft werden zu können, muss das zu prüfende Objekt formalen Kriterien entsprechen, damit die Analyse angewendet werden kann. Compiler können zum Beispiel lediglich Code überprüfen. Zu beachten ist bei diesen Analysen, dass sie eine Vielzahl von Daten liefern, die möglichst automatisch vorverarbeitet werden sollten, um den Überblick zu bekommen. Diesen Überblick können Metriken, wie zum Beispiel die Anzahl unabhängiger Pfade in einem Programm als zyklomatische Zahl erfasst werden. Diese Metriken haben keine direkte Aussagekraft, können aber für Abschätzungen genutzt werden. So kann eine hohe zyklomatische Zahl darauf hinweisen, dass ein Programmteil besonders kompliziert ist und Reviews hier besonders wichtig sind. [53, S. 98 - 108]

Dynamischer Test

Dynamische Tests bezeichnen das Testen im engeren Sinne, bei dem ein Testobjekt ausgeführt wird. Damit das Testobjekt testbar ist, muss es lauffähig sein. Dies bedeutet im frühen Stadium der Entwicklung, dass wie oben erläutert, entweder Platzhalter, Testtreiber oder ein Backbone-System vorhanden sein müssen, die den Rahmen für einen Test bilden. Wichtig ist, dass die Testfälle für diesen Test systematisch entwickelt werden und das zu erwartende Verhalten vorher festgelegt wird. [53, S. 109 - 113]

Bei der Erstellung der Tests gibt es zwei grundlegende Verfahren, die gewählt werden können. Bei der Erstellung der Tests kann das Testobjekt entweder als White- oder als Blackbox angesehen werden. Wenn das Testobjekt als Whitebox angesehen wird, so werden Testfälle unter Berücksichtigung der Codebasis erstellt. Dies ist

allerdings zeitaufwändig und bei sich änderndem Code zudem nicht sinnvoll, wie oben begründet. Deshalb werden im Folgenden verschiedene Vorgehen der Testfallermittlung bei Blackboxtests vorgestellt. Bei diesen Tests wird angenommen, dass der Aufbau des Testobjekts nicht bekannt ist und Tests basierend auf der zugrundeliegenden Spezifikation erstellt. [53, S. 112 - 113]

Eine Möglichkeit Testfälle auszuwählen ist die Äquivalenzklassenbildung. Bei dieser Auswahl werden die Eingabedaten in Äquivalenzklassen eingeteilt. Zu einer Äquivalenzklasse gehören alle Daten, bei denen das Testobjekt sich gleich verhalten sollte. Dementsprechend ist jeweils ein Test mit einem Repräsentanten aus einer Äquivalenzklasse ausreichend, weil sich der Test bei den anderen Repräsentanten der Äquivalenzklasse gleich verhalten sollte. Neben Äquivalenzklassen für gültige Werte muss es ebenso Äquivalenzklassen für ungültige Werte geben. [53, S. 114 - 119]

Angenommen es gibt eine Funktion, die den Verkaufspreis für ein Auto festlegt und je nach Höhe des Verkaufspreise werden unterschiedliche Rabatte gewährt. Wenn es einen Rabatt bis 30.000 Euro gibt und einen Rabatt für mehr als 30.000 Euro, dann sind die gültigen Äquivalenzklassen Eingaben größer als 0 Euro und kleiner gleich 30.000 Euro sowie Eingaben von größer als 30.000 Euro bis zum maximalen Wert. Darüber hinaus sind ungültige Äquivalenzklassen Eingaben kleiner gleich 0 Euro oder falls möglich falsche Inputs wie null zu beachten. Aus jeder dieser Äquivalenzklassen werden Repräsentanten ausgewählt, um zu testen, also zum Beispiel 15.000 Euro, 35.000 Euro, -5.000 Euro und null. [53, S. 114 - 115]

Wenn mehrere Eingabeparameter vorhanden sind, so werden Kombinationen aus Repräsentanten aller gültigen Äquivalenzklassen gebildet, sowie jeweils ein Repräsentant einer ungültigen Äquivalenzklasse in Kombination mit Repräsentanten gültiger Äquivalenzklassen. Wenn zu viele Testfälle durch die Kombination der Repräsentanten gültiger Äquivalenzklassen entstehen, kann die Anzahl durch Priorisierung reduziert werden, wie zum Beispiel durch eine Reduktion nach Wahrscheinlichkeit der Eingabe oder Bevorzugung von Testfällen mit Grenzwerten.[53, S. 119 - 121]

Der Vorteil der Äquivalenzklassenbildung ist, dass keine unnötigen Testfälle zur Ausführung kommen. Allerdings werden möglicherweise zu wenig Testfälle ausgeführt, weil das Verfahren nicht fehlerorientiert arbeitet. Es werden lediglich zufällige Repräsentanten ausgewählt, obwohl Fehler häufig bei Repräsentanten auftreten, die an den Grenzbereichen der Äquivalenzklassen liegen. [53, S. 125]

Deshalb ist eine sinnvolle Ergänzung zur Äquivalenzklassenbildung die Grenzwertanalyse. Diese Analyse kann angewendet werden, wenn die Eingabedaten in Äquivalenzklassen unterteilt werden können und Grenzen bei diesen Testdaten festgelegt werden können. An jeder dieser Grenzen werden jeweils die Grenzwerte selbst sowie die daneben liegenden Werte getestet. Im obigen Beispiel ist 30.000 Euro ein solcher Testfälle. Bei 30.000 Euro soll der Rabatt noch nicht gewährt werden. Bei 29.999 Euro soll der Rabatt ebenfalls nicht gewährt werden, während bei 30.001 Euro der Rabatt gewährt werden muss. Diese Grenzfälle können ebenfalls wie bei der Äquivalenzklassenbildung kombiniert und anschließend reduziert werden. [53, S. 125 -133]

Neben den Eingabedaten hat oft auch der Zustand des Systems einen Einfluss auf sein Verhalten. Diese Tatsache wird bei den zustandsbezogenen Tests berücksichtigt. Hier wird ermittelt, welche Zustände und Zustandsübergänge das System einnehmen kann. Anschließend wird ein Baum aus diesen Zuständen und Zustandsübergängen gebildet, indem jede mögliche Abfolge von Zuständen ohne Zyklen berücksichtigt wird. Aus diesem Baum können anschließend die Testfälle abgeleitet werden. Auch hier kann eine Reduktion notwendig werden. So kann eine Reduktion auf, jeder Zustand und jeder Zustandsübergang muss mindestens einmal erreicht bzw. ausgeführt werden, erfolgen. [53, S. 133 - 141]

Bei den bisherigen Verfahren zur Testfallbestimmung wurden die Eingabeparameter als unabhängig angesehen. Die Tatsache, das die Parameter einen wechselseitigen Einfluss haben können, wird bei der Entscheidungstabellentechnik berücksichtigt. Hier werden mögliche Ursachen und mögliche Wirkungen in eine Tabelle aufgenommen, um daraus Testfälle zu generieren. Ein Beispiel dafür sind als Ursachen die Gültigkeit einer Bankkarte und die Korrektheit eines PINs. Eine Spalte in der Entscheidungstabelle und damit ein Testfall sind die Ursachen *Bankkarte ist gültig* und *PIN ist korrekt* sowie die Wirkung, dass der Benutzer Zugriff auf ein Auswahlmenü bekommt. Durch dieses Verfahren können bisher unbeachtete Testfälle berücksichtigt werden, allerdings ist die Erstellung und Reduktion der Entscheidungstabelle sehr zeitaufwändig. [53, S. 141 - 145]

Eine weitere Möglichkeit zur Testfallermittlung ist anwendungsfallsbasiert. Bei dieser Ermittlung werden aus dem Anwendungsfalldiagramme die Anwendungsfälle extrahiert und zu jedem dieser Anwendungsfälle muss es mindestens einen Testfall geben. Diese Möglichkeit empfiehlt sich insbesondere zur Bestimmung von Testfällen für den Systemtest. [53, S. 145 - 148]

2.6.2.3 Testprozess

Unter Testprozess wird hier nach dem Prozessbegriff aus Abschnitt 2.5.2.3 der Mehrwert überprüfte Qualität verstanden. Um diesen Mehrwert zu schaffen und die Ressourcen dafür zu beschaffen sind verschiedene Tätigkeiten notwendig. Diese Tätigkeiten werden zu dem Prozess Testmanagement oder kurz Testprozess zusammengefasst. Der Testprozess des Projekts ist noch in Arbeit, wird sich aber an dem Testprozess aus Abbildung 2.13 anlehnen.

2.6.2.4 Testmanagement

Testmanagement bezeichnet die Aufgabe, Tests zu planen, durchzuführen bzw. die Durchführung zu begleiten und auch auszuwerten. [53, S. 10] Die Planung von Tests umfasst zunächst konzeptuelle Tätigkeiten, wie das Erstellen einer Testpolitik, eines Testkonzepts und von Testplänen sowie das Erarbeiten und Überarbeiten eines Testprozesses. Darüber hinaus müssen die notwendigen Ressourcen für den Testprozess beschafft und die Notwendigkeit der Test begründet werden. Auch die Auswahl und

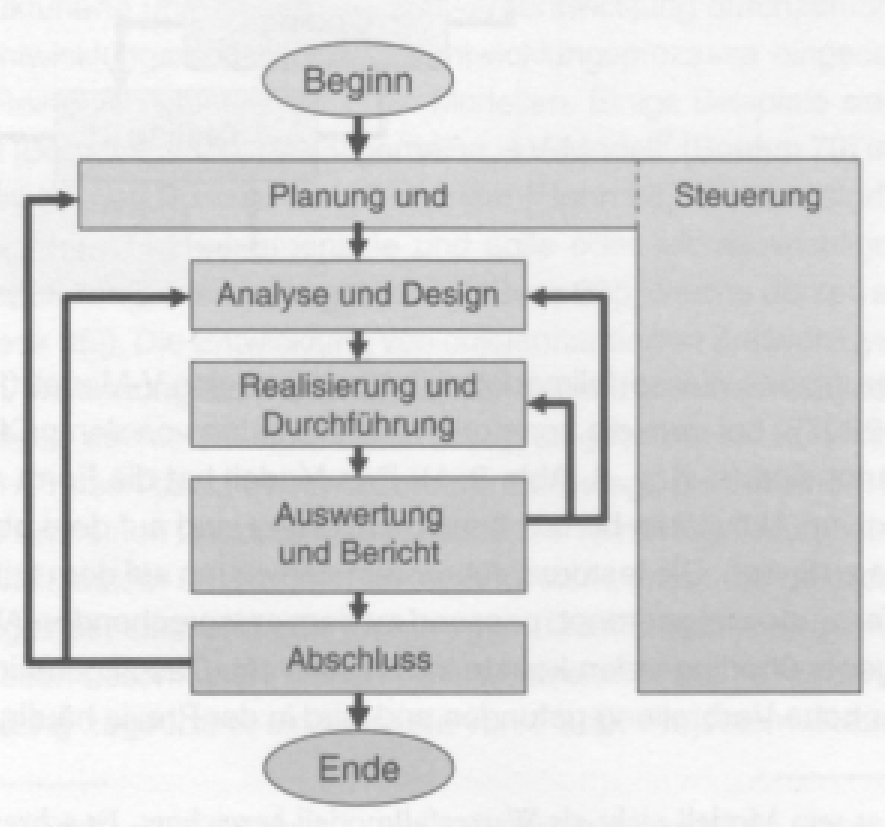


Abb. 2.12 Vorgehensmodell Testprozess [53, S. 20]

Einführung von Werkzeugen zur Unterstützung des Testprozesses werden vom Testmanagement übernommen. Schließlich umfasst die Aufgabe des Testmanagements auch das Erstellen und Kommunizieren von Testergebnissen. [53, S. 173]

2.6.2.5 Testpolitik

Die Dokumentation ist ein wesentlicher Bestandteil von Tests. Diese Dokumentation setzt sich aus verschiedenen Dokumenten zusammen, die hierarchisch geordnet werden können, wie in Abbildung 2.13 verdeutlicht wird. In der Testpolitik wird definiert, was unter dem Begriff Testen verstanden wird, durch welchen Prozess das Testen umgesetzt wird und welche Vorgaben und Ziele für den Testprozess gelten. In einem Unternehmen ist dies ein Dokument, das unternehmensweite Gültigkeit besitzt. In diesem Projekt entspricht die Testpolitik genau diesem Abschnitt 2.6. [54, S. 49 - 55]

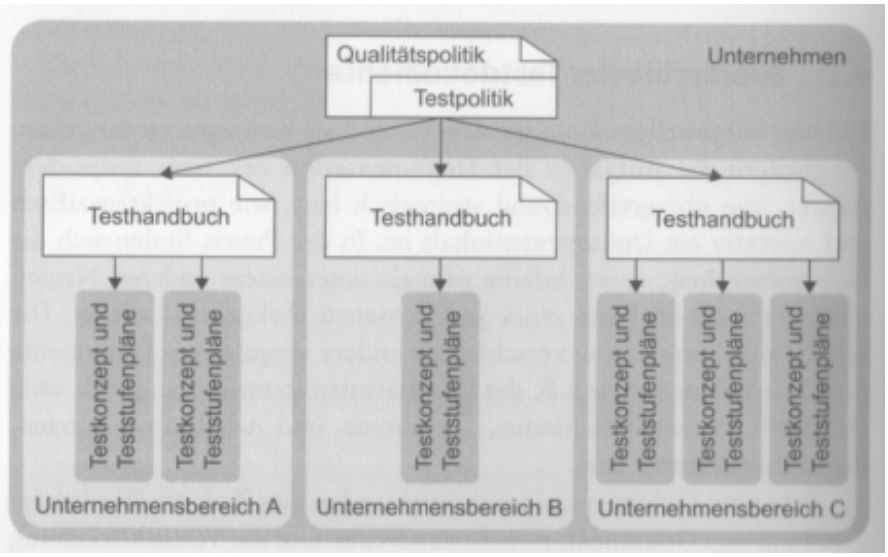


Abb. 2.13 Dokumente des Testmanagements [54, S. 50]

2.6.2.6 Testhandbuch

Im Testhandbuch wird die grundlegende Teststrategie festgelegt, indem bestimmt Fragestellungen hinsichtlich Tests beantwortet werden. Im Testhandbuch wird zunächst festgehalten, auf welche Art und Weise aus den Risiken eines Projekt die Testaktivitäten abgeleitet werden. Darüber hinaus wird auch bestimmt, in welcher Umgebung Tests durchgeführt werden, wann sie ausführt werden und wie sie automatisiert werden können. Zudem finden sich im Testhandbuch Aussagen dazu, welche Teststufen berücksichtigt werden sollten und wann in welchem Projektstadium welche Stufe angewendet werden soll. Im Testhandbuch wird auch die Kontrolle des Testmanagement näher erläutert, wie zum Beispiel auf welche Art und Weise werden die Testaktivitäten und der Einsatz der Ressourcen überwacht werden. Zusätzlich findet sich im Testhandbuch auch die Auskunft darüber, welches Integrationsverfahren verwendet wird. [54, S. 55 - 58]

2.6.2.7 Testkonzept

Im Testkonzept findet sich die konkrete Umsetzung des Testhandbuches für ein Projekt. Hier werden konkrete Vorgaben darüber getroffen, welche Eingangskriterien Voraussetzung für die verschiedenen Teststufen sind und unter welchen Ausgangskriterien die Tests als abgeschlossen gelten. Neben allgemeinen Zielen finden sich hier auch die wünschenswerten Testfälle für das System. Zudem werden die Umsetzung dieser Ziele konkret geplant und Ressourcen zu gewiesen. Abschließend wird

im Testkonzept auch festgehalten, auf welche Art und Weise die Berichte über den Testprozess verfasst werden. [54, S. 59 - 63]

2.6.3 Umsetzung

Die Realisierung der Tests musste eingeschränkt werden, da die Kapazitäten für das Testen aller Komponenten nicht gegeben waren. Unit Tests gestalteten sich als problematisch, da praktisch alle Systeme auf Daten aus anderen Systemen angewiesen sind und es wäre sehr aufwändig entsprechende Mock Ups zu erstellen und zu pflegen.

Es wurden folgende Maßnahmen zur Sicherstellung der Qualität ergriffen. Zunächst wurden Änderungen an der Software auf ihre Kompilierbarkeit geprüft. Damit dies automatisch geschieht wurde Jenkins verwendet, wie in Abschnitt 2.4 beschrieben. Dies sorgt für stets kompilierbaren Code im Repository. Des Weiteren wurden zu jedem Release Systemtests durchgeführt. Das erwartete Verhalten entsprach den Zielen der jeweiligen Version (siehe Abschnitt 7), die Überprüfung der internen Zustände erfolgte mittels Datenbankausgabe, Log oder via dem Debugmodus von Visual Studio. Ergebnis dieser Systemtests war spätestens nach Fehlerfindung, Erkennung und Behebung eine Übereinstimmung zwischen dem Verhalten des Systems und dem Verhalten aus den gesetzten Zielen der jeweiligen Version.

Abschließend wurde das Verhalten des Systems ausführlich in der Evaluation untersucht. Diese ist in Abschnitt 9 beschrieben.

Das Ergebnis dieser Maßnahmen ist eine hohe Zuversicht in der Funktionsfähigkeit unseres Systems.

Kapitel 3
Problemanalyse

Ziel der Problemanalyse ist es, dass die an der Projektgruppe beteiligten Parteien, die Studenten und die Betreuer, die gleichzeitig als Kunden fungieren, zu einer gemeinsamen Vorstellung des Projekts gelangen. Dazu sollen innerhalb der Problemanalyse die Akteure und die auftretenden Systeme identifiziert werden. Des Weiteren sollen die Anwendungsfälle gemeinsam erarbeitet werden, um daraufhin in Kapitel 4 die Anforderungen an das zu entwickelnde System exakt definieren zu können.

Im folgendem Abschnitt wird zunächst das bestehende Problem erläutert, indem die Frage geklärt wird, welche Bedingungen gegeben sind und in welchem Rahmen das Projekt entwickelt wird. Daraufhin wird das Vorgehen der Problemanalyse beschrieben. Sie gibt Antwort auf die Fragen, wie die Analyse durchgeführt wurde und welche Erhebungstechniken eingesetzt worden sind. Im letzten Abschnitt werden die Ergebnisse der Problemanalyse dokumentiert. Sie beinhalten die identifizierten Akteure und Systeme, sowie die ermittelten Anwendungsfälle und deren Beschreibung.

3.1 Problembeschreibung

Das Thema für die Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“ sieht vor, dass ein System entwickelt wird, welches die Steuerung von Energieerzeugern, -speichern und -verbrauchern eines simulierten Gebäudes in einer simulierten Umgebung realisiert. Als Energieerzeuger sollen Blockheizkraftwerke und Photovoltaikanlagen simuliert werden, als Speicher dienen simulierte Batterie- und Wärmespeicher. Die simulierte Umgebung ist durch Wetterdaten definiert, die beispielsweise Einfluss auf die Energieerzeugung der PV-Anlage und so Auswirkungen auf die gesamte Steuerung der Energieerzeuger, -speicher und -verbraucher haben können.

Das Problem besteht zum einen in der Entwicklung des Steuersystems, zum anderen in der Simulation des Gebäudes um das Steuersystem anwenden und testen zu können. Des Weiteren muss das genaue Ziel der Steuerung und der Umfang beziehungsweise der Detailgrad der Simulation definiert werden.

3.2 Vorgehen

Die Projektgruppe Hardwarebasierte Simulation energieautonomer Gebäude sah sich zunächst vor der Entscheidung, in welcher Form die Problemanalyse stattfinden sollte. Zur näheren Auswahl standen die Anfertigung eines Interviews oder die Gestaltung eines Workshops. Ein Workshop bietet den Vorteil, dass alle beteiligten Parteien zusammengebracht werden und so eine schnelle Abstimmung untereinander ermöglicht wird. Mangelnde Fachkenntnisse sprachen zudem gegen die Anfertigung eines Interviews. Die Gruppe entschied sich daher für die Erhebungstechnik Workshop.

Der Workshop wurde in zwei Sitzungen aufgeteilt. In der ersten Sitzung am 10.06.2014 wurde zunächst gemeinsam eine mit Zweigen vorgefertigte Mindmap erarbeitet. Diese bildete die Grundlage für die Vision (siehe Kapitel 1.1) und die weitere Problemanalyse. Darauffolgend wurde ein Brainstorming mit den Betreuern durchgeführt, welches das Ziel verfolgte, die Grundlage für die weitere Anforderungsdokumentation zu legen. Zum einen sollten alle teilnehmenden Akteure und Komponenten eindeutig identifiziert, zum anderen Anwendungsfälle für das zu entwickelnde System definiert werden. Dazu wurden vorgefertigte Schablonen verteilt, die durch die Kunden einzeln beschrieben wurden um möglichst variantenreiche Ergebnisse zu erhalten. Die Ergebnisse wurden anschließend diskutiert und zur Vermeidung von Redundanz gefiltert.

Der zweite Teil des Workshops fand in der darauffolgenden Woche am 17.06.2014 statt. Hier wurde weiter über die verschiedenen Anwendungsfälle diskutiert, diese den identifizierten Akteuren zugeordnet und dann zu Gruppen gebündelt.

Im weiteren Verlauf der Problemanalyse wurden darauf aufbauend Gespräche mit den Kunden geführt um Details zu klären und zusätzliche Informationen über die Kundenvorstellungen zu erhalten.

Die Ergebnisse der Problemanalyse werden in den folgenden Abschnitten dokumentiert. Sie bilden den Grundstein für die Identifizierung und Beschreibung der Anforderungen.

3.3 Ergebnis

Das Ergebnis der Problemanalyse ist die eindeutige Formulierung des Ziels für die Projektgruppe. Es soll ein EMS für ein aus mehreren Komponenten bestehendes SF entwickelt werden, für welches das EMS eine Eigenverbrauchsoptimierung durch-

führt. Die Komponenten des SF, im weiteren als EVS-Komponenten bezeichnet, bestehen aus Energieerzeugern, -speichern und -verbrauchern als Teil eines simulierten Gebäudes. Die Energieerzeuger zur Versorgung des Gebäudes mit elektrischer Energie sind Photovoltaikanlagen und Blockheizkraftwerke, die zusätzlich thermische Energie erzeugen. Zur Speicherung stehen elektrische und thermische Speicher zur Verfügung. Die Energieverbraucher sind unterteilt in nicht-steuerbare Verbraucher, die die Grundlast des Gebäudes bilden, und steuerbare Verbraucher (zum Beispiel Klima- oder Belüftungsanlage). Vorausgesetzt wird, dass es sich bei dem simulierten Gebäude um unterschiedliche Gebäude hinsichtlich des Zwecks und der Größe handeln kann. Es soll lediglich durch einen hohen Wärme- und Strombedarf, und steuerbare Verbraucher gekennzeichnet sein. In Frage kommen unter anderem die Simulation von Krankenhaus-, Schul- oder Hotelgebäuden. Die Gebäude müssen unter frei wählbaren Wetter- und Klimabedingungen analysierbar sein. Auch sollen individuell weitere Energieverbraucher innerhalb des Gebäudes manuell hinzugefügt oder entfernt werden können.

Für das EMS und SF sollen beliebige Szenarien integriert werden können. Ein solches Szenario beinhaltet die Kenndaten des definierten Gebäudes, also Grundlast bezüglich Strom- und Wärmebedarfs, Eigenschaften über die steuerbaren Verbraucher und die dezentralen Energieerzeuger und -speicher in der Simulation. Dazu gehört unter anderem die Dimensionierung der Energieanlagen, das heißt Angaben darüber, wie viel elektrische und thermische Energie durch die Anlagen bereitgestellt werden können. Des Weiteren werden zwei Wetterdatenquellen für das Szenario verwendet. Die erste besteht aus Vorhersagedaten, welche zur Berechnung von Einspeisedaten und Lastprognosen genutzt werden. Die zweite Quelle liefert reelle Wetterdaten, die von den Vorhersagen abweichen können.

Mit dem Start der Simulation soll durch das entwickelte Energiemanagementsystem ein Ablaufplan, der sogenannte Fahrplan, für die Energieversorgung berechnet werden, der in 15 minütiger Taktung in Abhängigkeit von veränderten Last- beziehungsweise Wetterdaten aktualisiert beziehungsweise angepasst wird. Ziel der Fahrplanerstellung ist die Eigenverbrauchsoptimierung zur Minimierung der Energiebezugskosten im Rahmen der definierten Simulationsdauer. Das Ergebnis soll hinsichtlich der Fragen evaluiert werden können, was das Energiemanagement im Bezug auf ein Szenario bewirkt habe, ob dem öffentlichen Stromnetz mehr Energie im Vergleich zum ungesteuerten Ablaufplan entnommen werden musste oder welche Parameter beziehungsweise Konfiguration im Vergleich verschiedener Szenarien einen positiven Einfluss auf das erzielte Ergebnis hatte.

Die weiteren Ergebnisse der Problemanalyse sind in den folgenden Abschnitten dokumentiert. Die Anwendungsfalldiagramme und deren Beschreibungen veranschaulichen die Benutzerinteraktionen mit den zu entwickelnden Systemen EMS und SF. Außerdem werden die verschiedenen Akteure und die einzelnen Systeme, sowie deren abstrakte Zusammenhänge, definiert.

3.3.1 Anwendungsfalldiagramm

Auf Grundlage des durchgeführten Workshops mit den Mitgliedern und Betreuern wurden die Anwendungsfälle bzw. Use-Case (deutsch: Anwendungsfall) (UC) erarbeitet und in zwei Diagrammen festgehalten. Sie verdeutlichen, dass das zu entwickelnde Produkt mindestens aus zwei Systemen bestehen wird.

Das ist zum Einen das Energiemanagementsystem, welches je nach Optimierungsziel den Tagesverlauf der Energieversorgungssystem (EVS)-Komponenten optimieren soll, und zum Anderen das Simulationsframework, anhand dessen ein Gebäude mit seinen Komponenten simuliert werden soll und das zum Testen und Evaluieren des EMS benötigt wird.
Die Evaluation soll dabei anhand eines bestimmten Szenarios geschehen, das vom EMS optimiert und vom SF simuliert wird. Das EMS soll beim Durchführen seiner Funktion jedoch nicht erkennen, ob es in einem realen oder einem simulierten Szenario eingesetzt wird. Daher wurde auch eine strikte Trennung innerhalb der Diagramme für die Anwendungsfälle des EMS und SF vorgenommen.

Eine genaue Beschreibung der Anwendungsfälle für die beiden Hauptbestandteile des Systems ist in Kapitel 3.3.5 gegeben.

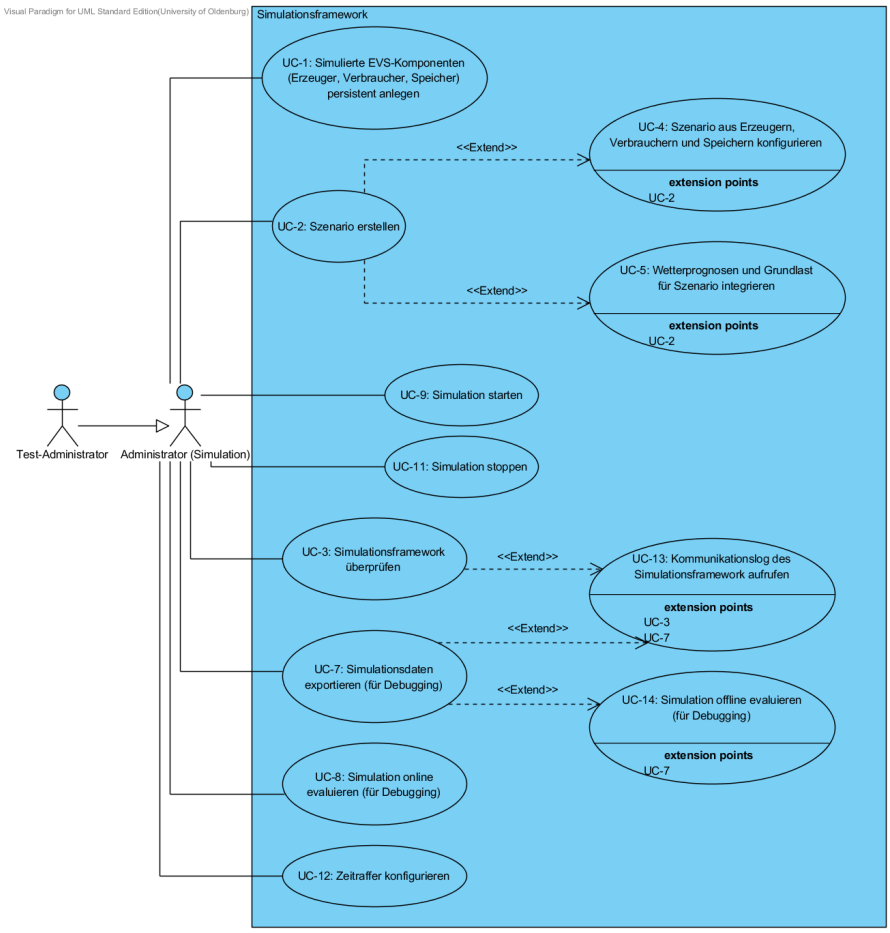


Abb. 3.1 UseCase-Diagramm zum Simulationsframework

Für das Simulationsframework sind zwei Akteure als Nutzer des Systems vorgesehen. Der Akteur Administrator(Simulation) kann alle Funktionen des SF ausführen. Zusätzlich gibt es einen Test-Administrator, der ebenfalls alle Funktionen ausführen kann. Der Unterschied zwischen diesen beiden Akteuren liegt darin, dass der Test-Administrator zusätzlich vollen Zugriff auf die Funktionen des EMS hat und so das Gesamtsystem ausführen und testen kann. Damit erhält er alle Möglichkeiten zum Debuggen und Evaluieren des Gesamtsystems.

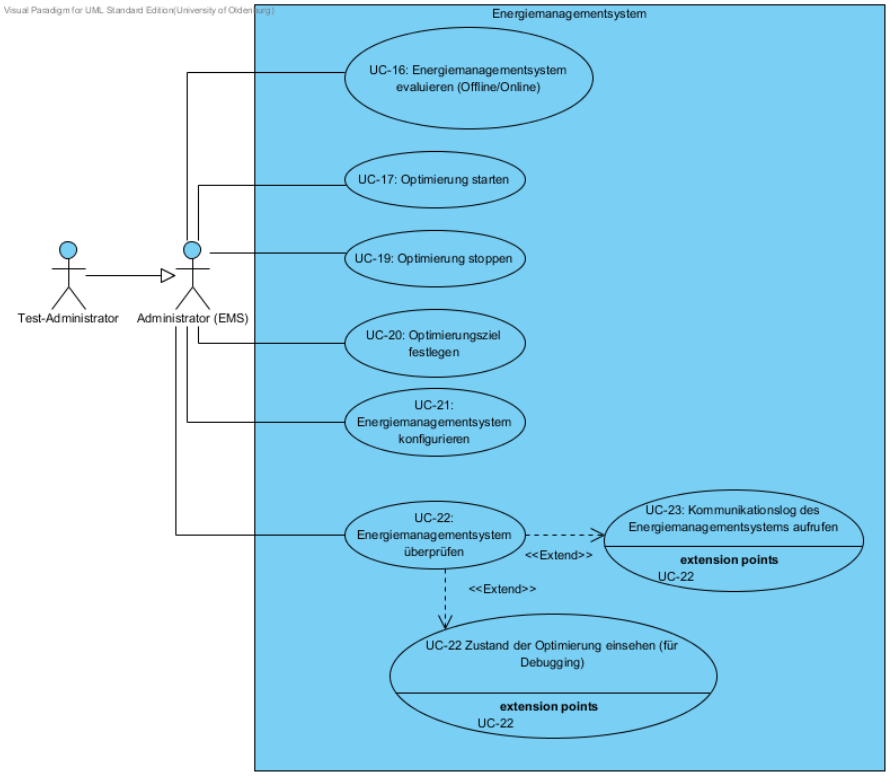


Abb. 3.2 UseCase-Diagramm zum Energiemanagementsystem (EMS)

Der Administrator (EMS) erhält Zugriff auf alle möglichen Funktionen des Energiemanagementsystems. Damit kann er das System sowohl ausführen, als auch debuggen. Im realen Einsatz des Systems ohne Simulation wäre er der eigentliche Nutzer des Systems. Wie im Anwendungsfalldiagramm zum Simulationsframework ersichtlich ist der zweite Akteur der Test-Administrator, der ebenfalls vollständigen Zugriff auf das System erhält.

Im EMS stand ein dritter Akteur zur Diskussion, der als Anwender des Energiemanagementsystems lediglich auf Teile des zu entwickelnden Systems zugreifen hätte können. Ein Beispiel für einen solchen einfachen Anwender wäre der Facility Manager eines Gebäudes, der in seinen Zugriffsrechten beschränkt wäre. Da die Frage nach den Anforderung für diesen einfachen Anwender nicht vollends geklärt wurde, ist dieser Akteur zum aktuellen Zeitpunkt für das System nicht relevant. Jedoch soll das System so aufgebaut werden, dass es Anwender mit beschränkten Zugriffsrechten geben kann. Die im Diagramm erwähnten Akteure werden einzeln im Kapitel 3.3.2 genau beschrieben.

3.3.2 Akteure

Die Akteure sind beteiligte Personen, die sich außerhalb des beschriebenen Systems befinden und als Benutzer des Systems gelten. In diesem Abschnitt werden die für das Projekt identifizierten Akteure detailliert beschrieben.

Akteur	Administrator (EMS)
Beschreibung	Der Administrator konfiguriert und wartet das Energiemanagementsystem (EMS). Die Konfiguration erfordert eine einfache und intuitive Bedienbarkeit des Systems. Zur Wartung benötigt er Einblick in Log-Dateien, die Informationen über den Systemzustand und der Kommunikation beinhalten. Gleichzeitig ist er Anwender des EMS und möchte die Steuerung mehrerer EVS-Komponenten in einem Gebäude optimieren. Damit er als Anwender den Erfolg und Stand der Optimierung verfolgen kann, muss er den Tagesverlauf der Optimierung und den Zustand der EVS-Komponenten einsehen können. Der Administrator besitzt Fachkenntnisse zu den EVS-Komponenten und technische Kenntnisse über das EMS und die Konfiguration des Systems.

Akteur	Administrator (Simulation)
Beschreibung	Der Administrator des Simulationsframeworks erstellt das Szenario, für welches das Energiemanagementsystem (EMS) getestet werden soll. Um das Szenerio zu erstellen konfiguriert er die simulierten EVS-Komponenten und pflegt alle weiteren benötigten Daten ein. Hierfür wird eine einfache und intuitiv nutzbare Benutzeroberfläche benötigt. Des Weiteren wartet er das Simulationsframework. Dazu benötigt er einen Einblick in die Log-Dateien, die Daten zum Systemzustand, zur Kommunikation und der simulierten EVS-Komponenten.

Akteur	Test-Administrator
Beschreibung	Der Test-Administrator hat Zugriff auf den gesamten Umfang an Funktionalitäten, die das System bereitstellt. Er testet das gesamte System, um die Korrektheit und die Erfüllung der Anforderungen zu gewährleisten.

3.3.3 Systemdiagramm

Die beiden herausgestellten Systeme sind das Energiemanagementsystem und das Simulationsframework. Zusätzlich wurde im Rahmen der Problemanalyse als zusätzliche Komponente die Steuerungsebene identifiziert, die den jeweiligen Akteuren als Benutzerschnittstelle für das EMS und/oder SF dient.

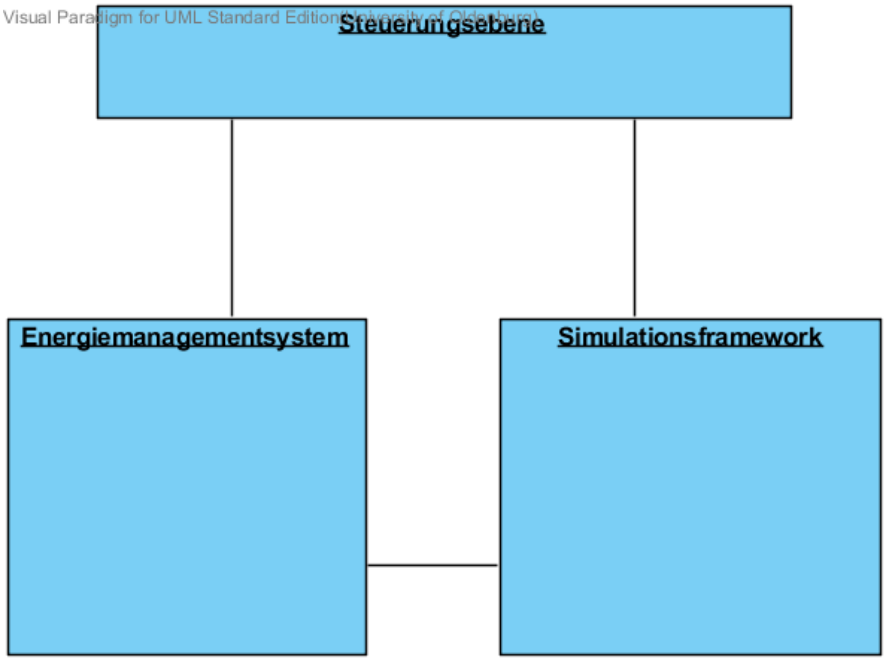


Abb. 3.3 Abstrakte Abbildung des Gesamtsystems

Der Grund für die Aufteilung in die drei Systeme (siehe Abbildung 3.3) liegt darin, dass das EMS das eigentliche Produkt der Projektgruppe werden soll, welches unabhängig vom SF ausführbar und in der Realität für Gebäude einsetzbar ist. Das SF bietet die Testumgebung für das entwickelte EMS und kann idealerweise auch für andere Projekte verwendet werden. Die SE bildet die Schnittstelle für den Anwender und den zwei Systemen EMS und SF. Sie ermöglicht das Konfigurieren der beiden Systeme und koordiniert das Starten und Stoppen der Optimierung und der Simulation. Über die Steuerungsebene soll der Zeitraffer für die Simulation des Szenarios eingestellt werden und die Visualisierung der Zustände des EMS und SF realisiert werden.

3.3.4 Systeme

Die Systeme sind im Kontext dieses Projekts die logisch abgegrenzten Teilaspekte des Projekts, die eine eigenständige Aufgabe übernehmen. Das Projekt besteht aus den drei Systemen Energiemanagementsystem, Simulationsframework und Steuerungsebene.

System	Energiemanagementsystem
Beschreibung	Das Energiemanagementsystem ist ein System, dass das Zusammenspiel von Erzeugern, Verbrauchern und Energiespeichern hinsichtlich einer Zielfunktion über einen festgelegten Zeitraum optimiert. Das EMS enthält insbesondere das Optimierungssystem und die Auswertungskomponente. Das System kommuniziert mit den Komponenten über Steuersignale und Zustände. Für die Optimierung benötigt es Prognosen und aktuelle Daten. Das EMS soll so konzipiert werden, dass es für in der Realität existierende Gebäude, die EVS-Komponenten enthalten, eingesetzt werden kann.

System	Simulationsframework
Beschreibung	Das Simulationsframework dient in erster Linie dem Testen des Energiemanagementsystems. Es simuliert möglichst realitätsnah das Gebäude mit EVS-Komponenten, seiner Grundlast und die Umgebung, welche durch aktuelle Wetterdaten und Wetterprognosen definiert ist.

System	Steuerungsebene
Beschreibung	Die Steuerungsebene bildet die Schnittstelle zwischen den Anwendern und den zwei Systemen Energiemanagementsystem und Simulationsframework. Die Steuerungsebene ermöglicht das Konfigurieren der beiden Komponenten und koordiniert das Starten und Stoppen der Optimierung und Simulation, sowie die Einstellung für den Zeitraffer. Sie ermöglicht zusätzlich die Visualisierung der Zustände des Energiemanagementsystems und der Simulation.

3.3.5 Anwendungsfallbeschreibung

In diesem Abschnitt werden die in den Diagrammen dargestellten Anwendungsfälle detailliert beschrieben, um ein klares Verständnis des Systems zu gewährleisten. Durch die tabellenartige Darstellung sind weitere Informationen, die für den einzelnen Anwendungsfall wichtig sind, dokumentiert. Nummer und Name sorgen für eine eindeutige Identifizierung. Die Beschreibung erörtert, was in dem Anwendungsfall

passiert. Des Weiteren werden die beteiligten Akteure und Komponenten, sowie verwendete Anwendungsfälle aufgezeigt. Die Vorbedingung beschreibt die Bedingungen, die erfüllt sein müssen, damit der jeweilige Anwendungsfall ausgeführt werden kann. Der Zustand, der nach einem erfolgreichen Durchlauf des Anwendungsfall erwartet wird, ist mit der Nachbedingung dokumentiert. Die Anwendungsfälle UC-1 bis einschließlich UC-14 beziehen sich auf das Simulationsframework. Die darauffolgenden Anwendungsfälle sind dem Energiemanagementsystem zugeordnet.

Anwendungsfall		*Anwendungsfallnr.*
Simulierte EVS-Komponenten (Erzeuger, Verbraucher, Speicher) persistent anlegen		UC-1
Beschreibung	Der Administrator (Simulation) legt eine simulierte EVS-Komponente persistent an und fügt sie dem Gerätepool hinzu. Anlegen bedeutet hier, dass der Typ der EVS-Komponente ausgewählt wird, ihre Kennzahlen festgelegt werden und eine eindeutige Kennzeichnung zugewiesen bekommt.	
Akteure	Administrator (Simulation)	
Komponenten	Simulationsframework	
verwendet		
Vorbedingung	Eingabemaske für EVS-Komponenten wurde geöffnet und EVS-Komponententypen stehen zur Verfügung.	
Nachbedingung	Eine EVS-Komponente ist dem Gerätepool hinzugefügt worden und persistent gespeichert.	

Anwendungsfall		*Anwendungsfallnr.*
Szenario erstellen		UC-2
Beschreibung	Der Administrator (Simulation) erstellt ein Szenario für die Simulation mit einem Titel.	
Akteure	Administrator (Simulation)	
Komponenten	Simulationsframework	
verwendet		
Vorbedingung	Der Administrator (Simulation) hat das Simulationsframework gestartet. Das System ist bereit.	
Nachbedingung	Das Szenario ist erstellt und kann konfiguriert werden.	

Anwendungsfall	*Anwendungsfallnr.*
Simulationsframework überprüfen	UC-3
Beschreibung	Der Administrator (Simulation) muss den Ablauf des Simulationsframeworks und die Funktionalität der Komponenten überprüfen können.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-11
Vorbedingung	Ein Szenario wurde gewählt und die Simulation wurde gestartet.
Nachbedingung	Der Administrator (Simulation) hat benötigte Informationen erhalten.

Anwendungsfall	*Anwendungsfallnr.*
Szenario aus Erzeugern, Verbrauchern und Speichern konfigurieren	UC-4
Beschreibung	Der Administrator (Simulation) konfiguriert ein Szenario aus Erzeugern, Verbrauchern und Speichern, indem er EVS-Komponenten dem Szenario hinzufügt oder entfernt.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-2
Vorbedingung	Szenario wurde erstellt.
Nachbedingung	EVS-Komponenten wurden dem Szenario hinzugefügt.

Anwendungsfall	*Anwendungsfallnr.*
Wetterprognosedaten und Grundlast für Szenario integrieren	UC-5
Beschreibung	Der Administrator (Simulation) wählt Wetterprognosedaten und die Grundlast des Gebäudes, welches simuliert werden soll, für ein Szenario aus. Mit den Wetterprognosedaten wird im Simulationsframework das reelle Wetter berechnet, welches von den Prognosen abweichen kann. Mit Hilfe der Lastgangsdaten kann das Energiemanagementsystem die Lastprognose für das simulierte Gebäude berechnen.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-2
Vorbedingung	Das Szenario ist erstellt und kann konfiguriert werden.
Nachbedingung	Die Wetterprognosedaten und Grundlast des zu simulierenden Gebäudes sind ausgewählt.

Anwendungsfall	*Anwendungsfallnr.*
Prognosedaten für Wetter und Lastgang konfigurieren	UC-6

Beschreibung	Der Administrator (Simulation) konfiguriert die Abweichung der Prognosedaten für Wetter- und Lastgang von den realen Daten. Die Prognosedaten werden benötigt, damit das Energiemanagementsystem (EMS) den Fahrplan für die Geräte erstellen kann. Die reellen Wetterdaten und der reelle Lastgang können von den Prognosen abweichen und zu Änderungen im Fahrplan führen.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-2
Vorbedingung	Das Szenario ist erstellt und kann konfiguriert werden.
Nachbedingung	Die Prognosedaten für Wetter und Lastgang wurden konfiguriert.

Anwendungsfall	*Anwendungsfallnr.*
Simulationsdaten exportieren (für Debugging)	UC-7

Beschreibung	Der Administrator (Simulation) exportiert Simulationsdaten zum Debugging nach einer Simulation.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-11
Vorbedingung	Die Simulation wurde gestoppt.
Nachbedingung	Die Simulationsdaten sind exportiert.

Anwendungsfall	*Anwendungsfallnr.*
Simulation online evaluieren (für Debugging)	UC-8

Beschreibung	Der Administrator (Simulation) evaluiert zur Laufzeit der Simulation die Daten. Enthalten in den Daten sind unter anderem die EVS-Komponentendaten, Wetterdaten und Prognosen und Kommunikationsdaten.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	
Vorbedingung	Die Simulation läuft.
Nachbedingung	

Anwendungsfall	*Anwendungsfallnr.*
Simulation starten	UC-9
Beschreibung	Der Administrator (Simulation) startet die Simulation.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	
Vorbedingung	Das Szenario für die Simulation wurde vollständig konfiguriert. Das System ist bereit.
Nachbedingung	Die Simulation wurde gestartet.

Anwendungsfall	*Anwendungsfallnr.*
Simulation stoppen	UC-11
Beschreibung	Der Administrator (Simulation) stoppt die Simulation.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	
Vorbedingung	Das Szenario für die Simulation wurde vollständig konfiguriert. Die Simulation läuft.
Nachbedingung	Die Simulation wurde gestoppt.

Anwendungsfall	*Anwendungsfallnr.*
Zeitraffer konfigurieren	UC-12
Beschreibung	Der Administrator (Simulation) konfiguriert den Zeitraffer vor der Simulation.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	
Vorbedingung	Die Simulation ist noch nicht gestartet.
Nachbedingung	Zeitraffer ist konfiguriert.

Anwendungsfall	*Anwendungsfallnr.*
Kommunikationslog des Simulationsframeworks aufrufen	UC-13
Beschreibung	Der Administrator (Simulation) kann zur Überprüfung der Simulation das Kommunikationslog der gesamten Simulation aufrufen. Das Kommunikationslog wird während der Simulation angelegt.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-3
Vorbedingung	Die Simulation wurde gestartet und ein Kommunikationslog angelegt.
Nachbedingung	Der Kommunikationslog ist geöffnet.

Anwendungsfall	*Anwendungsfallnr.*
Simulation offline evaluieren (für Debugging)	UC-14
Beschreibung	Der Administrator (Simulation) evaluiert die exportierten Simulationsdaten offline.
Akteure	Administrator (Simulation)
Komponenten	Simulationsframework
verwendet	UC-7
Vorbedingung	Die Simulationsdaten wurden exportiert.
Nachbedingung	

Anwendungsfall	*Anwendungsfallnr.*
Energiemanagementsystem evaluieren	UC-16
Beschreibung	Der Administrator (EMS) evaluiert das Energiemanagementsystem. Er soll die Einsparung gemäß dem Optimierungsziel mit dem ungesteuerten Verbrauch vergleichen können.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Optimierung wurde gestartet. Die Dauer der Optimierung genügt für einen Vergleich.
Nachbedingung	Der Administrator (EMS) hat Kenntnis gewonnen über die Effizienz der Optimierung. Das EMS arbeitet weiterhin.

Anwendungsfall	*Anwendungsfallnr.*
Optimierung starten	UC-17
Beschreibung	Der Administrator (EMS) startet die Optimierung.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Die Optimierung ist betriebsbereit.
Nachbedingung	Die Optimierung wurde gestartet. Die EVS-Komponenten erhalten optimierende Steuersignale.

Anwendungsfall	*Anwendungsfallnr.*
Optimierung stoppen	UC-19
Beschreibung	Der Administrator (EMS) stoppt die Optimierung.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Die Optimierung wurde gestartet.
Nachbedingung	Die Optimierung ist gestoppt. Die EVS-Komponenten arbeiten ohne optimierende Steuersignale weiter.

Anwendungsfall	*Anwendungsfallnr.*
Optimierungsziel festlegen	UC-20
Beschreibung	Der Administrator (EMS) legt ein Optimierungsziel fest. Dies könnte beispielsweise Kostenoptimierung, Eigenverbrauchsoptimierung oder weitere Optimierungsarten enthalten.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Das EMS wurde noch nicht gestartet.
Nachbedingung	Das EMS kann gemäß dem gewählten Optimierungsziel gestartet werden.

Anwendungsfall	*Anwendungsfallnr.*
Energiemanagementsystem konfigurieren	UC-21
Beschreibung	Der Administrator (EMS) muss das Energiemanagementsystem konfigurieren können. Durch die Konfiguration kann er die für das EMS notwendigen Daten der EVS-Komponenten einpflegen, die vom EMS gesteuert werden sollen.
Akteure	Administrator (Optimierung)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Das EMS wurde korrekt installiert.
Nachbedingung	Das EMS kann gestartet werden.

Anwendungsfall	*Anwendungsfallnr.*
Energiemanagementsystem überprüfen	UC-22
Beschreibung	Der Administrator (EMS) muss das Energiemanagementsystem überprüfen können. Er kontrolliert den Systemzustand der Optimierung und der EVS-Komponenten. Zum Systemzustand der Optimierung gehört beispielsweise der momentane Fahrplan und der Tagesverlauf bis zum aktuellen Zeitpunkt.
Akteure	Administrator (Optimierung)
Komponenten	Energiemanagementsystem
verwendet	
Vorbedingung	Das EMS ist korrekt konfiguriert und gestartet. Der erste Tagesplan ist erstellt. Die Optimierung hat die ersten Zustände der EVS-Komponenten erhalten.
Nachbedingung	Die zur Überprüfung des EMS notwendigen Informationen wurden dem Administrator (EMS) zur Verfügung gestellt. Das EMS arbeitet weiterhin.

Anwendungsfall	*Anwendungsfallnr.*
Kommunikationslog des Energiemanagementsystems aufrufen	UC-23
Beschreibung	Der Administrator (EMS) muss das Kommunikationslog des Energiemanagementsystem (EMS) einsehen können, um Fehlerzustände erkennen und analysieren zu können.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	UC-19
Vorbedingung	Das EMS wurde gestartet und ein Kommunikationslog wurde angelegt.
Nachbedingung	Der Kommunikationslog ist geöffnet.

Anwendungsfall	*Anwendungsfallnr.*
Zustand der Optimierung einsehen (für Debugging)	UC-24

Beschreibung	Der Administrator (EMS) muss den Zustand der Optimierung einsehen können, um Fehlerzustände erkennen und analysieren zu können. Das umfasst den bisherigen Verlauf und den aktuellen Fahrplan.
Akteure	Administrator (EMS)
Komponenten	Energiemanagementsystem
verwendet	UC-19
Vorbedingung	Das EMS wurde gestartet.
Nachbedingung	Die Informationen über den Zustand der Optimierung wurden dem Administrator (EMS) zur Verfügung gestellt.

Kapitel 4
Anforderungen

Eine Anforderung ist eine Aussage über eine zu erfüllende Eigenschaft oder zu erbringende Leistung des Produkts, welches im Rahmen der Projektgruppe entwickelt wird. Mit dem zu entwickelnden Produkt ist nicht nur das implementierte System gemeint, sondern es umfasst sowohl Software und Hardware als auch Abnahmekriterien, Handbuch, Zwischenbericht, Abschlussbericht und weiteres, was dem Auftraggeber zum Abschluss des Projekts ausgehändigt werden muss. Zu allen Teilen des Produkts können daher Anforderungen gestellt und formuliert werden.

Es war Aufgabe der Projektgruppe in Zusammenarbeit mit dem Auftraggeber diese Anforderungen zu identifizieren und dokumentieren. In Abschnitt 4.1 wird auf die Vorgehensweise der Anforderungserhebung eingegangen und kurz die Erhebungsvariante erläutert. Anschließend sind die Stakeholder in Abschnitt 4.2 beschrieben. Zunächst gibt es eine Grobstruktur des Gesamtsystems, um die Anforderungen daran besser einordnen zu können. Beginnend mit Abschnitt 4.3 werden die Anforderungen detailliert beschrieben.
Die wesentlichen Anforderungen an die drei Hauptkomponenten des Systems (EMS, SF, SE) werden damit aufgelistet. Abschließend werden in Abschnitt 4.4 Prozessanforderungen dargestellt. Diese beinhalten z.B. Anforderungen an die Lieferbestandteile des Projektes.

Mit dem Übergang vom Zwischenbericht zum Abschlussbericht wurden sämtliche Anforderungen aktualisiert oder neu verfasst. Da die Projektgruppe bei der Art der Optimierung die Wahl zwischen verschieden Lösungen hatte, waren die Anforderungen im Zwischenbericht sehr grundsätzlich gehalten. Nach der Entscheidung für das Sampling (Abschnitt 5.7), konnten die Anforderungen in diese Richtung präzisiert und überflüssige oder nicht mehr aktuelle Anforderungen entfernt werden. Alle überarbeiteten Anforderungen wurden mit einem “A” in der Anforderungsnummer versehen.

Im Abschnitt 8 wurden alle Anforderungen bezüglich ihres Erfüllungsgrades untersucht und den Kategorien vollständig umgesetzt, teilweise umgesetzt und nicht umgesetzt zugeteilt. Außerdem wurde für jede Anforderung skizziert, auf welche

Weise sie umgesetzt wurde bzw. was die Gründe dafür waren, dass sie nicht umgesetzt werden konnten.

4.1 Vorgehen

Anforderungen bieten in einem Systementwicklungsprozess eine wichtige Basis für die Kommunikations-, Diskussions- und Argumentations- sowie Vertragsgrundlage. Ein gemeinsames Verständnis und Wissen soll erarbeitet und durch die Anforderungen widergespiegelt werden. Sie bilden in Projekten zum Einen die rechtliche Grundlage für die Ausschreibungen und Vertragsgestaltung. Zum Anderen basiert die spätere Abnahme des Systems auf Tests gemäß den Anforderungsspezifikationen. Ein maßgeblichen Einfluss haben sie zudem auf die Systemarchitektur und auch Erweiterungen und Änderungen geschehen auf Basis der dokumentierten Anforderungen.

Die Grundlage für die Erhebung unserer Anforderungen waren die zuvor erarbeitete Vision und die Anwendungsfälle (siehe Abschnitt 3.3.1). Eine Abdeckung aller Anwendungsfälle durch die Anforderungen sollte nach der Anforderungserhebung erreicht sein.

Um das Vorgehen der Anforderungserhebung möglichst erfolgreich zu gestalten wurden zu Beginn unterschiedliche Erhebungstechniken vom Anforderungsmanagement gesammelt und anschließend diskutiert, so dass ein projektspezifisches Verfahren angewendet werden konnte. Zur Auswahl der Erhebungstechniken standen Interviews, ein Anforderungsworkshop, ein Fragebogen, Snowcards und Kreativitätstechniken (siehe [50]). Andere Methoden wie die direkte Beobachtung , Selbstaufschreibung oder Mitarbeit (siehe [27, 50]) waren auf Grund der Projektmöglichkeiten oder mangelnden Fachkenntnissen bereits vornherein ausgeschlossen. Der Einsatz eines Fragebogens oder Interviews wurde vom Anforderungsmanagement schnell verworfen, da nicht in allen Bereichen des Projektes genug Wissen über die Thematik vorhanden war und so grundlegende Fragen wohl möglich nicht genannt worden wären. Gleichzeitig bestand die Gefahr viele unnötige Fragen mit einzubringen.

Kreativitätstechniken wie beispielsweise Brainstorming oder Methode 635 (siehe [24, 62]) sind besonders geeignet um ungewöhnliche oder neue Ideen zu entwickeln. Ein vernünftiger Rahmen ist eine wichtige Voraussetzung für Kreativitätstechniken um nicht vom Thema abzukommen. Durch ein Brainstorming bspw. können in kurzer Zeit viele Ideen aufgenommen werden und später mit mehreren Personen weiterentwickelt und vertieft oder verworfen werden.

In einem Anforderungsworkshop werden zunächst Anforderungen höherer Ebene (Kundenanforderungen / Projektziele) erarbeitet. Die Auftraggeber und Stakeholder können im Workshop ihr Fachwissen gut einbringen. Durch Moderation und einen festgelegten Ablauf unterliegt ein Workshop klaren Regeln.

Um grobe Anforderungen zu finden, eignet sich ebenfalls die Anwendung von Snowcards. Auf vorgedruckten Karten werden möglichst alle Informationen einer Anforderung erfasst. Hierzu werden zunächst alle gesammelten Anforderungen ein-

deutig nummeriert und kurz beschrieben um anschließend Abhängigkeiten unter den Anforderungen mithilfe der Nummern zu definieren. Im nächsten Schritt werden die Anforderungen in der Gruppe diskutiert und um fehlende Informationen erweitert und detailliert.

Wie in Abschnitt 3.2 beschrieben, ist dabei die Entscheidung auf den Anforderungsworkshop gefallen. Um möglichst alle Vorstellungen der Kunden zu erfassen, wurde innerhalb des Workshops auch das Brainstorming genutzt. Einzeln sollten die Auftraggeber Akteure und Anforderungen an das System auf vorgefertigten Schablonen notieren. Während eines Touch, Turn & Talk wurden die gesammelten Anforderungen vorgestellt und wenn nötig diskutiert.

Snowcards hätten im Anforderungsworkshop durchaus angewandt werden können und waren zwischenzeitlich auch angedacht. Aus Zeitmangel in den Workshops wurde diese Methode jedoch verworfen. Entstanden war eine erste stichwortartige Sammlung an essenziellen Anforderungen an das System, die zunächst mit einem Wandbild und später digitalisiert festgehalten wurden. Eine detaillierte Beschreibung der Anforderungen wurde von der Arbeitsgruppe selbst verfasst.

Daraufhin wurde ein Anforderungstemplate erstellt und gefüllt, um eine detaillierte Beschreibung der Anforderungen zu dokumentieren. Die im Workshop erfassten und nun dokumentierten Anforderungen wurden abschließend im Dialog mit den Auftraggebern nochmals geprüft, verfeinert und priorisiert.

4.2 Stakeholder

Als Stakeholder oder Projektbeteiligter wird eine Person oder Gruppe bezeichnet, die in irgendeiner Weise von dem System betroffen ist und daher ein berechtigtes Interesse an der Entwicklung vorzuweisen hat. Die Systementwicklung im Rahmen der Projektgruppe verfolgt das Ziel, dieses Interesse beziehungsweise die Ansprüche aller Stakeholder zu berücksichtigen. Zu den Stakeholdern gehören die in Kapitel 4 identifizierten Akteure (Administrator (EMS), Administrator (Simulation), Test-Administrator), wobei der Administrator des Energiemanagementsystems als eigentlicher Kunde der Projektgruppe betrachtet wird. Außerdem gehören noch weitere Personen/Gruppen, die im folgenden Abschnitt beschrieben werden, zu den Stakeholdern des zu entwickelnden Systems.

Stakeholder	Anlagenhersteller von PV-Anlagen
Beschreibung	Es soll ein Kommunikationsprotokoll für simulierte PV-Anlagen verwendet werden, welches in der Realität benutzt wird oder vergleichbar ist mit standardisierten Kommunikationsprotokollen.

Stakeholder	Anlagenhersteller von BHKW-Anlagen
Beschreibung	Es soll ein Kommunikationsprotokoll für simulierte BHKW-Anlagen verwendet werden, welches in der Realität benutzt wird oder vergleichbar ist mit standardisierten Kommunikationsprotokollen.

Stakeholder	Anlagenhersteller von Energiespeichern (Batterien)
Beschreibung	Es soll ein Kommunikationsprotokoll für simulierte Energiespeicher verwendet werden, welches in der Realität benutzt wird oder vergleichbar ist mit standardisierten Kommunikationsprotokollen.

Stakeholder	Gesetzgeber
Beschreibung	Der Gesetzgeber ist ebenfalls ein Stakeholder. Jedoch soll lediglich das EEG (Erneuerbare-Energie-Gesetz) Einfluss auf die zu entwickelnde Optimierung haben. Dieses Gesetz regelt die bevorzugte Einspeisung von Strom aus erneuerbaren Energiequellen ins Stromnetz und garantiert deren Erzeugern eine feste Einspeisevergütung.

Stakeholder	Anbieter von Wetterdaten
Beschreibung	Mit Hilfe einer Sampling-Methode sollen alle EVS-Komponenten eigene Fahrpläne erstellen können, die für die nächsten für 24 Stunden gültig sind. Dafür benötigt u.A. die Photovoltaikanlage Prognosedaten für Temperatur- und Sonnenstunden. Daher wird eine Quelle für Wetterdaten (für 24 Stunden in viertelstündiger Taktung) benötigt, die für das zu entwickelnde System verwendet werden kann. Das angebotene Datenformat der Wetterdaten muss unterstützt und eine Schnittstelle zum Importieren entwickelt werden. Wird eine Gebäude innerhalb Oldenburg simuliert, wäre beispielsweise das Dezernat 4, Gebäudemanagement der Universität Oldenburg, eine mögliche Datenquelle für aktuelle Wetterdaten.

Stakeholder	Beckhoff / SESA-Lab Experte
Beschreibung	Die Verwendung der Beckhoff Schaltschrank IPCs als Hardwarekomponenten ist verpflichtend für die Projektgruppe. Zur Einarbeitung und Verwendung der Hardware dienen Handbücher und Tutorials. Als Experte auf dem Gebiet steht uns außerdem Martine Büscher vom SESA-Lab (OFFIS) als Ansprechpartner bei Fragen während der Einarbeitung und Verwendung der Hardware zur Seite.

Stakeholder	Energieversorgungsunternehmen
Beschreibung	Der Energieversorger hat Einfluss auf das System, da er die Stromtarifmodelle für aus dem öffentlichen Netz bezogenen Strom vorgibt. Dieses Tarifmodell kann die Optimierung beeinflussen, wie beispielsweise ein Aufladen des Energiespeichers aufgrund eines günstigen Nachtstromtarifs. Der Energieversorger ist jedoch auf den zuführenden Faktor beschränkt. Die Vergütung des ins öffentliche Netz eingespeisten Stroms soll (zunächst) keine Berücksichtigung finden.

Stakeholder	Netzbetreiber (Verteilnetzbetreiber / Übertragungsnetzbetreiber)
Beschreibung	Der Netzbetreiber ist ebenfalls ein Stakeholder, weil er Strom aus erneuerbaren Energiequellen nach dem EEG abnehmen und vergüten muss (siehe Gesetzgeber).

Stakeholder	Auftraggeber
Beschreibung	Der Auftraggeber des Projekts sind die drei Betreuer, die als unsere Kunden fungieren.

4.3 Systemanforderungen

Die Systemanforderungen sind die Anforderungen an das Energiemanagementsystem, das Simulationsframework und die Steuerungsebene. Gekennzeichnet sind die Anforderungen durch einen Titel und eine eindeutige Anforderungsnummer. Des Weiteren gibt der Status Auskunft darüber, ob die Anforderungen noch bearbeitet wird oder ob sie bereits geschlossen ist. Ist die Anforderung geschlossen, so ist sie entweder bereits umgesetzt oder die Umsetzung nicht mehr notwendig bzw. machbar. Die Grunde dafür werden im Abschnitt 8 beschrieben. Die Priorität gibt an, wie wichtig die jeweilige Anforderungen für das Projekt ist. Zur Auswahl stehen die Prioritäten kritisch, unkritisch oder optional. In der Beschreibung ist die eigentliche Anforderung an die jeweiligen Systeme formuliert. Zusätzlich werden zu jeder Anforderung Einschränkungen, eine Begründung, Querbezüge zu anderen Anforderungen, verwendete Anwendungsfälle (vgl. Abschnitt 3.3.5, die beteiligten Komponenten, Akzeptanzkriterien und Kommentare dokumentiert. Der Einfachheit halber wird in den Systemanforderungen der jeweilige Akteur immer als Anwender bezeichnet.
Abbildung 4.1 zeigt grob die Grenzen der einzelnen Systeme. Diese Abbildung soll das Verständnis der folgenden Anforderungen erleichtern. Im Diagramm sind die Systeme ‚Energiemanagementsystem, Simulationsframework und Steuerungsebene, in Verbindung mit den EVS-Komponenten und der bereits angedachten zentralen Datenbank zu sehen. An den Verbindungen sind kurze Beschreibungen zu deren

Funktionalität. Dieses Diagramm ist nur als Orientierung gedacht und beinhaltet nur eine grobe Übersicht.

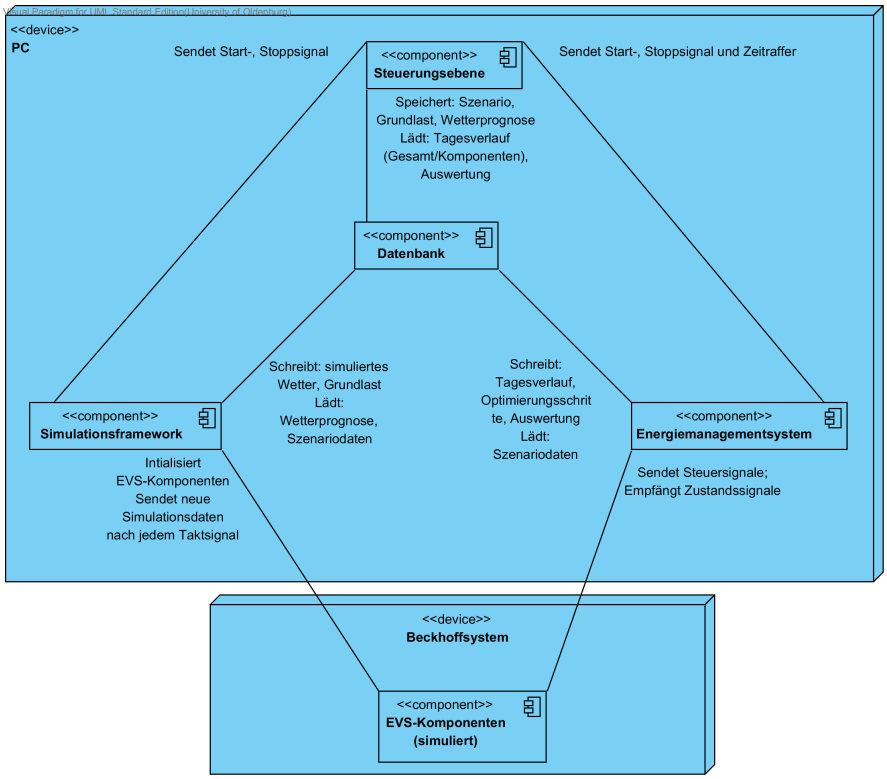


Abb. 4.1 Grober Überblick über das zu entwickelnde System

4.3.1 Energiemanagementsystem

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario (EMS) konfigurieren	EMS-10A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Möglichkeit bieten das Szenario für das Energiemanagementsystem zu konfigurieren. Außerdem muss eine Datenquelle für eine Grundlastprognose angegeben werden.		
Einschränkungen	Falls das Energiemanagementsystem für eine Simulation verwendet wird, ist die Dauer der Optimierung auf die Simulationsdauer beschränkt.		
Begründung	Das Energiemanagementsystem soll einem Szenario entsprechend konfiguriert werden.		
Querbezüge	–		
Anwendungsfälle	UC-21		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Das Energiemanagementsystem ist mit einem Szenario konfigurierbar und übernimmt bei der Konfiguration alle Einstellungen, die für den Betrieb des Energiemanagementsystem notwendig sind.		
Kommentare	Da alle EVS-Komponenten Sampling anbieten, entfällt eine Konfiguration des EMS mit diesen EVS-Komponenten.		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsziel festlegen	EMS-15A	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss die Möglichkeit bieten, das Optimierungsziel festzulegen. Das Optimierungsziel bestimmt auf welche Art die Optimierung aus den Samples der EVS-Komponenten einen Fahrplan auswählt.
Einschränkungen	–
Begründung	Das Energiemanagementsystem soll verschiedene Optimierungsziele verwalten können. Es soll modular aufgebaut sein, um es erweitern zu können.
Querbezüge	SE-EMS-25
Anwendungsfälle	UC-21
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Es muss mindestens ein Optimierungsziel auswählbar sein (Eigenverbrauchsoptimierung zur Minimierung der Energiebezugskosten). Es muss die Möglichkeit gegeben sein, weitere Optimierungsziele hinzufügen zu können.
Kommentare	Optional sollen weitere Optimierungsziele auswählbar sein.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlast (thermisch, elektrisch) abrufen	EMS-30A	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss Informationen über die Grundlast (thermisch, elektrisch) des Gebäudes abrufen können. Eine Schnittstelle für das Abrufen der Daten muss im EMS enthalten sein. Im Falle der Simulation gibt es eine Quelle für die Grundlast, die mit der Konfiguration des Szenarios integriert wird und über das Simulationsframework verfügbar ist.
Einschränkungen	–
Begründung	Die Grundlast wird für die Optimierung benötigt.
Querbezüge	–
Anwendungsfälle	–
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Die Grundlast steht der Optimierung zur Verfügung.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsschritt	EMS-60A	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss für die steuerbaren EVS-Komponenten anhand der Samples und gemäß dem gewählten Optimierungsziel den folgenden Betriebsschritt aller EVS-Komponenten bestimmen.
Einschränkungen	Ein Betriebsschritt gilt für das Optimierungsziel als optimal, wenn die Energiebezugskosten für eine vorher definierte Optimierungshorizont und vorher festgelegten Komponenten unter Berücksichtigung von Gerätebedingungen und zugrundeliegenden Umgebungsparametern, wie Wetter oder Grundlast des Gebäudes, minimal sind. Der berechnete Fahrplan muss nicht optimal aber möglichst gut sein. Eine Verwendung einer guten Heuristik wird durch diese Anforderung nicht ausgeschlossen. Die Unversehrtheit der Geräte darf unter keinen Umständen durch Nichtbeachtung der Gerätebedingungen gefährdet werden.
Begründung	Diese Anforderung ist ein Hauptbestandteil der Vision.
Querbezüge	EMS-70
Anwendungsfälle	–
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Der jeweils folgende Betriebsschritt für eine EVS-Komponente wurde gewählt. Dieser Betriebsschritt verletzt keine Gerätebedingungen, die zu einer Beschädigung der Geräte führen (z.B. Temperatur eines Kühlschranks über einen erlaubten Schwellenwert).
Kommentare	Die Samples einer EVS-Komponente stellen eine Auswahl möglicher Fahrpläne dieser EVS-Komponente dar. Die Samples, die den folgenden Betriebsschritt aller EVS-Komponenten vorgeben, bestimmen den aktuell verfolgten Fahrplan. Die Samples, die die ersten gewählten Betriebsschritte der EVS-Komponenten vorgeben, gelten als Initialplan.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Komponentenfahrplan anpassen	EMS-70	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss den Fahrplan für die steuerbaren EVS-Komponenten im vom Systemtakt vorgegebenen Abstand aktualisieren, d.h. neu optimieren.
Einschränkungen	–
Begründung	Die Umgebungsparameter, wie z.B. benötigte Heizwärme, Sonnenstand, Bewölkung etc. ändern sich mit der Zeit und sind nicht fehlerfrei vorhersehbar und müssen somit in eine aktuelle Berechnung einfließen.
Querbezüge	EMS-60A
Anwendungsfälle	–
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Der Fahrplan für alle steuerbaren EVS-Komponenten wird im vom Systemtakt vorgegebenen Abstand neu optimiert.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Steuerungssignale versenden	EMS-80A	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss die Wahl eines Fahrplans an die EVS-Komponenten übermitteln.
Einschränkungen	–
Begründung	Die EVS-Komponenten können nur nach dem Fahrplan agieren, wenn sie über diesen im Ganzen oder aufgeteilt in Steuersignalen informiert werden.
Querbezüge	EMS-60A, EMS-70
Anwendungsfälle	–
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Für die von uns benutzen EVS-Komponenten sind die nötigen Steuersignale vorhanden und können kommuniziert werden.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Ergebnis der Optimierung bereitstellen	EMS-90A	geschlossen	unkritisch
Beschreibung	Das Energiemanagementsystem muss das Ergebnis der Optimierung bereitstellen. Das Ergebnis der Optimierung kann dem Anwender mittels Visualisierungskomponente dargestellt werden und setzt sich als eine Kostenprognose für die nächsten 24 Stunden zusammen.		
Einschränkungen	–		
Begründung	Der Anwender möchte über das Ergebnis eines Optimierungsschrittes informiert werden. Veränderungen im Tagesverlauf im Vergleich zum Initialplan sollen ersichtlich sein, damit der Anwender die Funktionalität des EMS erkennen kann (auch zum Debugging).		
Querbezüge	EMS-60A, EMS-70		
Anwendungsfälle	UC-16		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Es ist nach jedem Systemtakt möglich, auf das Ergebnis der Optimierung zuzugreifen.		
Kommentare	Weitere Informationen zur Überprüfung des EMS sollen optional bereitgestellt werden.		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Samplings der EVS-Komponenten abrufen	EMS-100A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Samplings der EVS-Komponenten auslesen oder empfangen können.		
Einschränkungen	–		
Begründung	Die Optimierung benötigt die Samplings aller EVS-Komponenten um einen aktualisierten Fahrplan und damit den nächsten Betriebsschritt der EVS-Komponenten zu bestimmen.		
Querbezüge	EMS-60A, EMS-70		
Anwendungsfälle	UC-22		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Die Kommunikation der Daten zwischen EMS und EVS-Komponenten ist möglich.		
Kommentare	Es wird die Annahme getroffen, dass die EVS-Komponenten fehlerfrei und ohne Störung arbeiten. Es gibt keine Maßnahmen, um falsche oder unvollständige Samplings zu erkennen.		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsschritt bei Simulation vorzeitig beenden	EMS-120A	geschlossen	optional

Beschreibung	Das Energiemanagementsystem soll die Möglichkeit bieten, während einer Simulation einen Optimierungsschritt vorzeitig zu beenden.
Einschränkungen	–
Begründung	Um die Simulation zu beschleunigen und unnötig hohe Zeitspanne für kaum weitere Optimierung zu verhindern, kann das frühzeitige beenden eines Optimierungsschritt benutzt werden.
Querbezüge	EMS-60A, EMS-70, EMS-90A
Anwendungsfälle	–
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Die Optimierung im Energiemanagementsystem ermöglicht ein vorzeitiges beenden des aktuellen Optimierungsschrittes.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Samplings speichern	EMS-140A	geschlossen	unkritisch

Beschreibung	Das Energiemanagementsystem muss die gewählten Samplings zu jedem Zeitschritt speichern können.
Einschränkungen	–
Begründung	Um jeden Optimierungsschritt gegen den Initialplan offline evaluieren zu können, ist die Auswahl notwendig.
Querbezüge	SE-EMS-30
Anwendungsfälle	UC-16, UC-22
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Die gewählten Samplings zu jedem Zeitschritt sind gespeichert.
Kommentare	Die gewählten Samplings beinhalten die zu einem Zeitpunkt prognostizierten Werte für die Energieerzeugung bzw. den Energieverbrauch einer EVS-Komponente.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Interne Log erstellen	EMS-150	geschlossen	unkritisch
Beschreibung	Das Energiemanagementsystem muss eine Logdatei erstellen. Diese muss Informationen über Daten, Funktionsaufrufe und Fehler enthalten.		
Einschränkungen	Das interne Log und das Kommunikationslog können in eine Log-Datei geschrieben werden.		
Begründung	Für eine Bereinigung von Fehlern und die Wartung des Produkts wird ein Log benötigt, über den der Entwickler bzw. Administrator den Systemzustand einsehen kann, der zu den nicht gewünschten Ereignis geführt hat.		
Querbezüge	SE-EMS-05, EMS-160		
Anwendungsfälle	UC-16, UC-22		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Der Log enthält alle relevanten Daten ohne dabei zu viele redundante Informationen zu liefen.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Kommunikationslog erstellen	EMS-160	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss ein Kommunikationslog erstellen. Dieses dokumentiert die ein- und ausgehende Kommunikationen.		
Einschränkungen	Das interne Log und das Kommunikationslog können in eine Log-Datei geschrieben werden.		
Begründung	Für eine Bereinigung von Fehlern und die Wartung des Produkts wird ein Kommunikationslog benötigt, über den der Entwickler bzw. Administrator nachvollziehen kann, welches System, wann und was mit dem Energiemanagementsystem kommuniziert.		
Querbezüge	SE-EMS-05, EMS-150		
Anwendungsfälle	UC-23		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Der Log enthält alle relevanten Daten ohne dabei zu viele redundante Informationen zu liefen.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Logdatei des Energiemanagementsystems bereitstellen	EMS-180	geschlossen	kritisch

Beschreibung	Das Energiemanagementsystem muss Logdateien bereitstellen. Die Logdateien informieren über den Zustand und die Historie des Energiemanagementsystems.
Einschränkungen	–
Begründung	Der Anwender möchte Information über das Energiemanagementsystem erhalten um das Verhalten des EMS nachvollziehen zu können und zusätzlich ein Debugging durchzuführen.
Querbezüge	SE-EMS-05, EMS-150, EMS-160
Anwendungsfälle	UC-22
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Der Anwender hat Zugriff auf die Logdateien.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EMS beenden	EMS-170	geschlossen	kritisch

Beschreibung	Das EMS muss vom Anwender über die Steuerungsebene beendet werden können. Die EVS-Komponenten erhalten vom Energiemanagementsystem keine weiteren Steuersignale.
Einschränkungen	–
Begründung	Die Optimierung soll von der Steuerungsebene aus gesteuert werden können. Basierend darauf muss die Optimierung von der Steuerungsebene beendet werden können. Die EVS-Komponenten werden nicht mehr vom EMS gesteuert.
Querbezüge	SE-EMS-05
Anwendungsfälle	UC-19
beteiligte Komponente(n)	Energiemanagementsystem
Akzeptanzkriterien	Nach Beendigung der Optimierung arbeiten die simulierten EVS-Komponenten selbständig.
Kommentare	–

4.3.2 Simulationsframework

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Verrauschen der simulierten Wetterprognosedaten	SF-40A	geschlossen	kritisch

Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, die Wetterprognosedaten zu verrauschen. Damit soll sichergestellt werden, dass das simulierte Wetter von der Wetterprognose abweicht.
Einschränkungen	–
Begründung	Das simulierte Wetter muss aufgrund der geforderten Realitätsnähe von den integrierten Wetterprognosedaten abweichen.
Querbezüge	–
Anwendungsfälle	UC-6
beteiligte Komponente(n)	Simulationsframework, Steuerungsebene
Akzeptanzkriterien	Nach dem Verrauschen unterscheidet sich der Datensatz der Wetterprognose vom Datensatz des simulierten Wetters.
Kommentare	Die Eingabe des Anwenders wird über die Steuerungsebene vorgenommen. Das Verrauschen ist abhängig von einem Faktor.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Verrauschen der Grundlast	SF-55	geschlossen	kritisch

Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, die Grundlast zu verrauschen. Analog zum Verrauschen der Wetterprognosedaten, wird damit eine Abweichung der tatsächlichen thermischen und elektrischen Last des Gebäudes von der prognostizierten Grundlast ermöglicht.
Einschränkungen	Die Berechnung des Wärmebedarfs soll zunächst unabhängig von äußeren Faktoren sein (siehe Protokoll vom 16.09.14). Die Berechnung soll zunächst einfach gehalten werden und weitere Parameter können optional mit einbezogen werden (z.B. Temperatur).
Begründung	Der simulierte thermische und elektrische Energieverbrauch muss aufgrund der geforderten Realitätsnähe von prognostizierten Grundlast abweichen.
Querbezüge	SF-40A
Anwendungsfälle	UC-2, UC-5
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Nach dem Verrauschen unterscheidet sich der Datensatz der Grundlastprognose vom Datensatz der tatsächlichen Grundlast.
Kommentare	Die Eingabe des Anwenders wird über die Steuerungsebene vorgenommen. Mögliche weitere Parameter können vom Gebäudemodell abhängen: Gebäudegröße (Zimmer, Stockwerke), Standort, Nutzerverhalten und viele weitere. Die Berechnung kann daher sehr komplex werden. Das Verrauschen ist abhängig von einem Faktor.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario initialisieren (SF)	SF-60	geschlossen	kritisch

Beschreibung	Das Simulationsframework initialisiert das Szenario.
Einschränkungen	–
Begründung	Der Anwender möchte die Simulation starten. Dafür müssen die EVS-Komponenten angelegt und auf einen Initialzustand gesetzt werden. Nach der Initialisierung ist die Simulation startbereit.
Querbezüge	–
Anwendungsfälle	–
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Die Simulation kann gestartet werden.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation implizit starten	SF-70A	geschlossen	kritisch

Beschreibung	Das Simulationsframework startet bei *mit Simulation* konfiguriertem EMS ebenfalls die Simulation.
Einschränkungen	Beim Starten des EMS muss die Simulation implizit starten. Das eigenständige Starten der Simulation ist zunächst nicht vorgesehen, da in diesem Fall ein ungesteuerter Verbrauch simuliert würde.
Begründung	Der Anwender möchte das entwickelte EMS mit dem Szenario testen. Daher startet die Simulation implizit wenn das EMS gestartet wird.
Querbezüge	SF-60, SF-40A
Anwendungsfälle	UC-9
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Die Simulation wird automatisch gestartet, wenn das EMS mit Simulation konfiguriert wurde und startet.
Kommentare	Momentan ist der Taktgeber im EMS enthalten, daher besteht eine Abhängigkeit vom SF zum EMS. Es wurde die Möglichkeit gegeben, einen Taktgeber im Simulationsframework zu implementieren.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation separat starten	SF-80A	geschlossen	optional

Beschreibung	Das Simulationsframework bietet die Möglichkeit, die Simulation ohne das EMS zu starten.
Einschränkungen	SF-60, SF-40A
Begründung	Wenn der Anwender einen nicht optimierten Verlauf simulieren möchte, kann er die Simulation ohne das EMS starten (diese Möglichkeit ist in der Entwicklung nicht zwingend erforderlich).
Querbezüge	SF-170
Anwendungsfälle	UC-9
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Die Simulation ist ohne das EMS ausführbar.
Kommentare	Ein ungesteuerter Verbrauch wird nicht implementiert werden, da die Komponentenmodelle kein angemessenes Standard-Verhalten bieten.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EVS-Komponenten simulieren	SF-90	geschlossen	kritisch

Beschreibung	Das Simulationsframework muss die für das Szenario konfigurierten EVS-Komponenten simulieren.
Einschränkungen	Nur geforderte Komponenten (PV-Anlage, Batterie, BHKW und steuerbare Verbraucher) müssen simuliert werden können.
Begründung	Das EMS soll mit EVS-Komponenten die Optimierung durchführen.
Querbezüge	SF-60, SF-70A
Anwendungsfälle	–
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Alle Komponenten müssen ihren Ablauf simulieren können.
Kommentare	Auch mehrere Komponenten des selben Komponententyps sollen simuliert werden können.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Taktsignal empfangen	SF-110	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss ein Taktsignal empfangen können und führt auf Grund des Signals den folgenden Simulationsschritt durch.		
Einschränkungen	–		
Begründung	Durch Einsatz eines Zeitraffers und durch gleichmäßigen Ablauf von Simulation und EMS wird eine Clock eingesetzt. Diese sendet ein Taktsignal, welches empfangen werden muss.		
Querbezüge	SF-70A		
Anwendungsfälle	–		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Nach dem Empfangen eines Taktsignals wird der folgende Simulationsschritt ausgeführt.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Wetterprognosedaten und Wetter versenden	SF-130A	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, das simulierte Wetter sowie die Wetterprognose versenden zu können.		
Einschränkungen	–		
Begründung	Die PV-Anlage benötigt im Betrieb Informationen zum Wetter und beim Sampling Veränderungen des Wetters.		
Querbezüge	SF-40A		
Anwendungsfälle	–		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Das simulierte Wetter und die Wetterprognose stehen allen EVS-Komponenten, die diese Informationen benötigen, zur Verfügung.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlast und simulierte Wetter speichern	SF-135	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, den simulierten thermischen und elektrischen Energiebedarf des Gebäudes sowie das simulierte Wetter persistent zu speichern.		
Einschränkungen	–		
Begründung	Der Anwender kann die Simulation offline evaluieren		
Querbezüge	SF-40A, SF-55, SF-200		
Anwendungsfälle	–		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Der simulierte thermische und elektrische Energiebedarf des Gebäudes sowie das simulierte Wetter sind persistent gespeichert.		
Kommentare	Sinnvoll ist das Abspeichern in einer Datenbank.		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Kommunikationslog erstellen	SF-140	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss für die gesamte Kommunikation (vor und während der Simulation) ein Log erstellen.		
Einschränkungen	Das interne Logging und das Logging der Kommunikation kann in eine Datei geschrieben werden.		
Begründung	Damit ein Debugging und eine Evaluation der Kommunikation durchgeführt werden kann, soll ein Kommunikationslog angelegt werden.		
Querbezüge	SF-70A, SF-150		
Anwendungsfälle	–		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Der Log enthält alle relevanten Daten ohne dabei zu viele redundante Informationen zu liefen.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Internes Log erstellen	SF-150	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss ein systeminternes Log erstellen.		
Einschränkungen	Das interne Logging und das Logging der Kommunikation kann in eine Datei geschrieben werden.		
Begründung	Das systeminterne Logging dient dem Debugging.		
Querbezüge	SF-70A, SF-140		
Anwendungsfälle	–		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Der Log enthält alle relevanten Daten ohne dabei zu viele redundante Informationen zu liefen.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation implizit beenden	SF-160	geschlossen	optional
Beschreibung	Das Simulationsframework beendet die Simulation implizit, wenn das EMS beendet wurde.		
Einschränkungen	–		
Begründung	Beim Starten und Beenden des EMS wird die Simulation implizit gestartet beziehungsweise beendet. Da zunächst kein Ablauf mit ungesteuertem Verbrauch vorgesehen ist.		
Querbezüge	SF-70A		
Anwendungsfälle	UC-11		
beteiligte Komponente(n)	Simulationsframework		
Akzeptanzkriterien	Die Simulation wird automatisch beendet, wenn das EMS beendet wird.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation separat beenden	SF-170	geschlossen	optional

Beschreibung	Bei separatem Start der Simulation muss diese auch separat beendet werden können.
Einschränkungen	–
Begründung	Der separate Start der Simulation zum Simulieren des ungesteuerten Verbrauchs ist zunächst nicht vorgesehen. Wird dieses Feature implementiert, muss auch ein separates Stoppen der Simulation möglich sein.
Querbezüge	SF-80A
Anwendungsfälle	UC-11
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Der Anwender kann eine separat gestartete Simulation auch separat beenden.
Kommentare	Separates Stoppen darf nur bei ungesteuertem Verbrauch möglich sein. Separates Stoppen der Simulation während gesteuertem Verlauf durch das EMS ergibt keinen Sinn. Einzig ein Zusammenbruch der Kommunikation könnte damit simuliert werden.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Logdatei des Simulationsframework bereitstellen	SF-180	geschlossen	kritisch

Beschreibung	Das Simulationsframework muss Logdateien bereitstellen. Die Logdateien informieren über den Zustand und die Historie der Simulation und der in dem Szenario enthaltenen EVS-Komponenten.
Einschränkungen	–
Begründung	Der Anwender möchte Information über das Simulationsframework erhalten, um das Verhalten des EMS nachvollziehen zu können und zusätzlich ein Debugging des Simulationsframeworks durchzuführen.
Querbezüge	SF-140, SF-150
Anwendungsfälle	UC-3, UC-13
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Der Anwender hat Zugriff auf die Logdateien.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulationsdaten exportieren	SF-200	geschlossen	kritisch

Beschreibung	Das Simulationsframework muss die Möglichkeit geben, die Simulationsdaten zu exportieren. Die Simulationsdaten beinhalten die Simulationsergebnisse (Historie aller EVS-Komponenten und deren Zustände während der Simulation) und das Kommunikationslog der EVS-Komponenten.
Einschränkungen	–
Begründung	Der Anwender kann die Simulation offline evaluieren (Debugging, Test der Funktionalität des Systems).
Querbezüge	SF-190, SF-140, SF-150, SF-135
Anwendungsfälle	UC-7, UC-13, UC-14
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Der Anwender hat die Möglichkeit die Simulationsergebnisse und das Kommunikationslog zu exportieren.
Kommentare	–

4.3.3 Steuerungsebene

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EMS starten	SE-EMS-05	geschlossen	kritisch
Beschreibung	Das EMS muss vom Anwender über die Steuerungsebene gestartet werden können. Das Energiemanagementsystem wird mit den Daten des Szenarios initialisiert. Nach der Initialisierung kann die Optimierung starten.		
Einschränkungen	Das Starten der Optimierung mit einem simulierten Szenario startet automatisch auch die Simulation. Das Szenario für das Energiemanagementsystem wurde konfiguriert.		
Begründung	Die Optimierung soll von der Steuerungsebene aus gesteuert werden können. Basierend darauf muss der Start der Optimierung von der Steuerungsebene ausgeführt werden können.		
Querbezüge	–		
Anwendungsfälle	UC-17		
beteiligte Komponente(n)	Energiemanagementsystem		
Akzeptanzkriterien	Die Optimierung wurde gestartet.		
Kommentare	Beim Start der Optimierung müssen die EVS-Komponenten in einem konsistenten Zustand sein.		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsstatus visualisieren	SE-EMS-10A	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss den Optimierungsstatus des Energiemanagementsystems abrufen und visualisieren können.		
Einschränkungen	–		
Begründung	Der Anwender möchte über den Zustand der Optimierung informiert werden.		
Querbezüge	SE-EMS-15		
Anwendungsfälle	UC-15		
beteiligte Komponente(n)	–		
Akzeptanzkriterien	Es muss mindestens die Information visualisiert werden, ob die Optimierung läuft oder fertig ist.		
Kommentare	–		

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Statistik der Kostenprognose visualisieren	SE-EMS-15	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss das Ergebnis der Optimierung des Energiemanagementsystems abrufen und visualisieren können.
Einschränkungen	–
Begründung	Der Anwender möchte über eine Verbesserung bzw. eine Verschlechterung der Kostenprognose im Vergleich zum Initialplan informiert werden, um die Optimierung evaluieren zu können.
Querbezüge	SE-EMS-10A
Anwendungsfälle	UC-16
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Es muss wenigstens nach jedem Takt die berechnete Kostenprognose ausgegeben werden.
Kommentare	Da im 15-minuten Takt aktualisierte Fahrpläne erstellt werden, kann die aktuelle Kostenprognose optional gegen alle anderen Fahrpläne verglichen werden.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EVS-Komponentenzustände visualisieren	SE-EMS-20	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender Einsicht in den Status der EVS-Komponenten bieten. Der Zustand aller Verbraucher, Erzeuger und Speicher muss dargestellt werden. Die aktuellen Werte müssen den Anlagen spezifisch angezeigt werden (z.B. tabellarisch). Außerdem soll eine Historie über Planungs- und Störereignisse angezeigt werden.
Einschränkungen	Nicht regelbare Verbraucher werden nicht dargestellt. Nur ihr Energieverbrauch ist von Interesse.
Begründung	Der Anwender möchte bei Störungen im Ablauf informiert werden. Er möchte den Zustand der EVS-Komponenten gemäß dem geplanten Tagesverlauf überprüfen. Ausfälle sollen in der Steuerungsebene ersichtlich sein.
Querbezüge	–
Anwendungsfälle	UC-15
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Es muss mindestens der aktuelle Energiebedarf bzw. die aktuelle Energieproduktion sowie Störereignisse angezeigt werden.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsziel auswählen	SE-EMS-25	geschlossen	optional

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit bieten, das Optimierungsziel auszuwählen. Anschließend muss dem Energiemanagementsystem das gewählte Optimierungsziel mitgeteilt werden.
Einschränkungen	–
Begründung	Der Anwender soll die Möglichkeit haben, das gewünschte Optimierungsziel auszuwählen.
Querbezüge	EMS-15A
Anwendungsfälle	UC-15
beteiligte Komponente(n)	Steuerungsebene, Energiemanagementsystem
Akzeptanzkriterien	Es muss mindestens ein Optimierungsziel auswählbar sein.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Zeitraffer einstellen	SE-10	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit bieten, den Zeitraffer für die Ablaufgeschwindigkeit der Simulation und Optimierung einzustellen.
Einschränkungen	Es muss eine maximale Geschwindigkeit für den Zeitraffer geben, damit das EMS eine sinnvolle Optimierung gewährleisten kann. Während der Simulation soll der Zeitraffer nicht veränderbar sein.
Begründung	Es ist vorgegeben worden, dass sowohl die Simulation als auch die Optimierung schneller oder langsamer ablaufen können soll. Dafür muss auf der Steuerungsebene eine Bedienfläche erstellt werden, mit der ein Befehl für einen initialen Zeitraffer an die Simulation und das Energiemanagementsystem gesendet wird.
Querbezüge	–
Anwendungsfälle	UC-12
beteiligte Komponente(n)	Simulationsframework
Akzeptanzkriterien	Die Simulation und das EMS müssen weiterhin voll funktionsfähig sein. Die angegebene Geschwindigkeit soll mit dem ersten Taktsignal übernommen sein.
Kommentare	Die maximale Geschwindigkeit muss während der Implementierung evaluiert werden.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Gerätepool anlegen	SE-50	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, einen Gerätepool anzulegen. Dazu gehört das Hinzufügen, Ändern und Löschen von Geräten. Unter dem Begriff Geräte werden alle EVS-Komponenten verstanden. Die integrierten Geräte müssen mit Defaultwerten definiert und persistent gespeichert werden.
Einschränkungen	–
Begründung	Der Anwender möchte neue Geräte hinzufügen, wie beispielsweise ein modernes Blockheizkraftwerk, welches getestet werden soll. Bestehende Geräte können geändert oder gelöscht werden. Der Anwender soll dabei den Typ der EVS-Komponente auswählen, ihn mit Kennzahlen versehen und eine eindeutige Kennzeichnung zuweisen können. Aus dem Pool an Geräten soll der Anwender anschließend ein Szenario für das Simulationsframework konfigurieren können.
Querbezüge	–
Anwendungsfälle	UC-1
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Der Anwender kann aus den verschiedenen Typen der EVS-Komponenten ein Modell auswählen und dieses mit spezifischen Kennzahlen und einer Kennzeichnung versehen.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario (Simulation) konfigurieren	SE-60	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, ein Szenario für die Simulation zu konfigurieren. Die aus dem Gerätepool ausgewählten EVS-Komponenten müssen mit spezifischen Parameter und Initialwerten gefüllt werden. Zu den EVS-Komponenten gehören Energieerzeuger, -verbraucher und -speicher. Es muss eine Dauer für das Szenario eingegeben werden. Außerdem muss die Geschwindigkeit für den Zeitraffer angegeben werden.
Einschränkungen	–
Begründung	Der Anwender muss das Szenario konfigurieren können, damit im Simulationsframework eine Simulation gestartet werden kann.
Querbezüge	SE-50
Anwendungsfälle	UC-2, UC-4
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Der Anwender kann für ein Szenario EVS-Komponenten auswählen und die für diese Komponenten notwendigen Parameter übergeben. Er kann die Dauer für ein Szenario sowie die Geschwindigkeit für den Zeitraffer übergeben. Alle weiteren Parameter für die Definition eines Szenarios können entweder übergeben werden oder sind als Standardwerte verfügbar.
Kommentare	–

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Wetterprognosedaten dem Szenario (Simulation) hinzufügen	SE-70	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, einem konfigurierten Szenario Wetterprognosedaten hinzufügen zu können.
Einschränkungen	Die Wetterprognose muss Informationen enthalten über Temperatur und Strahlung in W/qm für (mindestens) alle 30 Minuten.
Begründung	Ohne Wetterprognosedaten kann die Simulation nicht starten. Es soll eine Schnittstelle zu einem Wetterdienst simuliert werden. Das Simulationsframework berechnet aus dem ausgewählten Wetterprognosedaten und einer durch den Anwender anzugebenden Abweichung das simulierte Wetter.
Querbezüge	SF-40A
Anwendungsfälle	UC-2, UC-5
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Der Anwender hat verschiedene Datensätze für die Wetterprognose zur Verfügung und kann einem Szenario einen Datensatz zuweisen.
Kommentare	Die Wetterprognosedaten müssen ausreichend lang für die Dauer der Simulation sein (12 Stunden Wetterdaten mehr als die Simulationsdauer).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlastmodell (Wärmebedarf, Strombedarf) dem Szenario (Simulation) hinzufügen	SE-80	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, dem Szenario ein Grundlastmodell hinzufügen zu können.
Einschränkungen	–
Begründung	Das Grundlastmodell beinhaltet prognostizierte Grundlastinformationen über den elektrischen und thermischen Energiebedarf des zu simulierenden Gebäudes. Auf dieser Basis wird die Grundlast des Gebäudes simuliert.
Querbezüge	SF-55
Anwendungsfälle	UC-2, UC-5
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Der Anwender hat verschiedene Datensätze für ein Grundlastmodell zur Verfügung und kann einem Szenario einen Datensatz zuweisen.
Kommentare	Referenzprofile zur Berechnung der Grundlast (VDI).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario persistent speichern	SE-90	geschlossen	kritisch

Beschreibung	Die Steuerungsebene muss ein Szenario persistent speichern können. Die Szenariodaten enthalten die ausgewählten EVS-Komponenten und deren Parameter. Bei Simulationsszenarien kommen die ausgewählten Wetterprognosedaten, Geschwindigkeit des Zeitraffers und das Grundlastmodell hinzu.
Einschränkungen	–
Begründung	Das Szenario soll später neu geladen werden können. Das Energiemanagementsystem kann auf diese Weise mit diesem Szenario erneut eine Optimierung durchführen.
Querbezüge	SF-70A
Anwendungsfälle	–
beteiligte Komponente(n)	Steuerungsebene
Akzeptanzkriterien	Ein konfiguriertes Szenario kann vom Anwender abgespeichert werden. Ein abgespeichertes Szenario kann vom Anwender geladen und erneut simuliert werden.
Kommentare	–

4.4 Prozess

4.4.1 Anforderungen an sonstige Lieferbestandteile

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht	D-ANF 10	offen
Beschreibung	Es muss ein Zwischenbericht abgegeben werden, der den Stand des Projekts zum Zeitpunkt der Abgabe darstellt.	
Querbezüge	D-ANF 20, D-ANF 30, D-ANF 40, D-ANF 50, D-ANF 60, D-ANF 70	
Kommentare	Die Abgabe des Zwischenberichts soll voraussichtlich Ende September bis Anfang Oktober erfolgen.	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Grundlagen	D-ANF 20	offen
Beschreibung	Der Zwischenbericht muss wissenschaftliches Hintergrundwissen bieten, welches die Grundlage für unsere Lösungsmethoden darstellt, wenn dieses nicht bereits im Seminarband abgedeckt wird.	
Querbezüge	D-ANF 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Projektvorgehen	D-ANF 30	offen
Beschreibung	Der Zwischenbericht muss das Vorgehen innerhalb des Projekts aufzeigen.	
Querbezüge	D-ANF 10	
Kommentare	Zum Projektvorgehen gehören unter anderem Organisation, Meilensteine, Projektstruktur und Rollen.	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Vorgehen der Anforderungsanalyse	D-ANF 40	offen
Beschreibung	Im Zwischenbericht muss das Vorgehen bei der Anforderungsanalyse inklusive wichtiger Zwischenergebnisse beschrieben werden.	
Querbezüge	D-ANF 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Anforderungen	D-ANF 50	offen
Beschreibung	Der Zwischenbericht muss alle aufgestellten Anforderungen beinhalten. Darunter werden sowohl Anforderungen an das Projekt, als auch Anforderungen, die im Rahmen des Projektumfeldes erhoben wurden, verstanden.	
Querbezüge	D-ANF 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Softwareentwurf	D-ANF 60	offen
Beschreibung	Der Zwischenbericht muss den Stand des Softwareentwurfs zum Zeitpunkt der Abgabe beinhalten.	
Querbezüge	D-ANF 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Zwischenbericht: Seminarband	D-ANF 70	offen
Beschreibung	Der Zwischenbericht muss den Seminarband beinhalten, der die in der Seminarphase abgegebenen Seminarausarbeitungen umfasst.	
Querbezüge	D-ANF 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht	D-ANF 80	offen
Beschreibung	Es muss ein Abschlussbericht am Ende der Projektphase abgegeben werden, der den Stand des fertigen Projekts darstellt.	
Querbezüge	D-ANF 90, D-ANF 100, D-ANF 110, D-ANF 120, D-ANF 130, D-ANF 140, D-ANF 150	
Kommentare	Die Abgabe des Abschlussberichts erfolgt voraussichtlich am Ende der vorlesungsfreien Zeit im Wintersemester 2014/2015.	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Inhalte des Zwischenberichts	D-ANF 90	offen
Beschreibung	Der Abschlussbericht muss analog zum Zwischenbericht Grundlagen, Vorgehen im Projekt, Vorgehen der Anforderungsanalyse, Anforderungen, Softwareentwurf und Seminarband beinhalten.	
Querbezüge	D-ANF 20, D-ANF 30, D-ANF 50, D-ANF 60, D-ANF 70, D-ANF 80	
Kommentare	Die Inhalte müssen dem aktuellsten Stand entsprechen.	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Benutzerhandbuch	D-ANF 100	offen
Beschreibung	Der Abschlussbericht muss ein Benutzerhandbuch enthalten, das die Installation, Konfiguration und Benutzung des Systems erläutert.	
Querbezüge	D-ANF 80	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Evaluation	D-ANF 110	offen
Beschreibung	Der Abschlussbericht muss die Ergebnisse und das Vorgehen der Evaluation beinhalten.	
Querbezüge	D-ANF 80	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Realisierung	D-ANF 120	offen
Beschreibung	Der Abschlussbericht muss die Realisierung des Produkts darstellen.	
Querbezüge	D-ANF 80	
Kommentare	Zur Realisierung gehören verwendete Werkzeuge, Implementierungsdetails (wo angebracht) und die Realisierung der Tests.	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Fazit	D-ANF 130	offen
Beschreibung	Der Abschlussbericht muss ein Fazit beinhalten, in dem eine reflektierende Beurteilung des Projekts erfolgt.	
Querbezüge	D-ANF 80	
Kommentare	Das Fazit muss folgende Fragen beantworten: Wie bewerten wir unser Projekt insgesamt rückblickend? Welche Entscheidungen/Methoden haben sich besonders ausgezahlt, welche waren eher hinderlich?	

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Offene Punkte	D-ANF 140	offen
Beschreibung	Der Abschlussbericht muss offene Punkte des Projekts aufzeigen, so dass sichtbar ist, welche Anforderungen und aus welchem Grund diese Anforderungen nicht umgesetzt wurden.	
Querbezüge	D-ANF 80	
Kommentare		

Titel	*Anforderungsnr.*	*Status*
Abschlussbericht: Anknüpfungspunkte	D-ANF 150	offen
Beschreibung	Der Abschlussbericht muss Stellen nennen, an denen das entwickelte Gesamtsystem am ehesten weiterentwickelt werden kann bzw. soll.	
Querbezüge	D-ANF 80	
Kommentare		

Titel	*Anforderungsnr.*	*Status*
Angabe von Autoren	D-ANF 160	offen
Beschreibung	Abschnitte in den Berichten müssen mit den Autorennamen gekennzeichnet werden.	
Querbezüge	D-ANF 10, D-ANF 80	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Version ohne Autorennamen	D-ANF 170	offen
Beschreibung	Es muss zusätzlich zur Version mit Autorennamen eine Version des Abschlussberichts ohne Autorennamen bereitgestellt werden.	
Querbezüge	D-ANF 80, D-ANF 160	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Layout	D-ANF 180	offen
Beschreibung	Die Berichte müssen ein lesbares Layout haben.	
Querbezüge	D-ANF 10, D-ANF 80	
Kommentare	Ansonsten ist das Layout der Berichte frei wählbar.	

Titel	*Anforderungsnr.*	*Status*
Seminarfolien	D-ANF 150	offen
Beschreibung	Die Seminarfolien sind kein geforderter Bestandteil der Berichte.	
Querbezüge	D-ANF 10, D-ANF 80	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Release	L-ANF 10	offen
Beschreibung	Am Ende des Projekts müssen die erstellten Systeme inklusive verwendeter Fremdbibliotheken in einem finalen Release abgegeben werden.	
Querbezüge	–	
Kommentare	Die Abgabe kann über einen Datenträger, einer URL oder als finale Version des Git-Repository erfolgen.	

Titel	*Anforderungsnr.*	*Status*
Readme	L-ANF 20	offen
Beschreibung	Das Release muss eine Readme beinhalten, die den Start der Software verständlich beschreibt.	
Querbezüge	L-ANF 10	
Kommentare	–	

4.4.2 Rechtliche-vertragliche Anforderungen

Titel	*Anforderungsnr.*	*Status*
Lieferung der vereinbarten Leistungen	D-ANF-RV 10	offen
Beschreibung	Die in Abschnitt 4.4.1 genannten Lieferbestandteile müssen erbracht werden. Weiterhin muss die Software zum vereinbarten Zeitpunkt abgegeben werden.	
Querbezüge	D-ANF 10, D-ANF 80	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Vergütung des Projekts	D-ANF-RV 20	offen
Beschreibung	Die Lieferung der vereinbarten Leistungen in Verbindung mit dem erfolgreichen Bestehen der Projektgruppe wird mit 24 Kreditpunkten vergütet. Eine monetäre Vergütung des Projektes erfolgt nicht.	
Querbezüge	D-ANF-RV 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Lizenzrechtliche Bestimmungen	D-ANF-RV 30	offen
Beschreibung	Wird Fremdsoftware verwendet, sind lizenzrechtliche Bestimmungen sowie gesetzliche Vorgaben einzuhalten. Eine Verwendung der Software nach Beendigung der Projektgruppe durch Dritte muss möglich sein. Bei Verwendung von Literatur sind die verwendeten Quellen anzugeben sowie die Rechtmäßigkeit der Nutzung zu überprüfen.	
Querbezüge	D-ANF-RV 10	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Wöchentliche Sitzungen	D-ANF-RV 40	offen
Beschreibung	Es finden im wöchentlichen Rhythmus Sitzungen statt. Für diese Sitzungen wird jeweils - in abwechselnder Form - ein Moderator und ein Protokollant bestimmt. Wesentliche Aspekte der Sitzungen (z.B. Vereinbarungen) sind zu protokollieren. Es herrscht Anwesenheitspflicht. Krankheit und genehmigter Urlaub befreien von der Anwesenheitspflicht.	
Querbezüge		
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Getroffene Vereinbarungen	D-ANF-RV 50	offen
Beschreibung	Mit den Betreuern getroffene Vereinbarungen sind einzuhalten. Vereinbarungen sind gültig, sobald sie in ein Sitzungsprotokoll aufgenommen wurden und dieses Protokoll in einer weiteren Sitzung abgenommen wurde, d.h. keine Einwände mehr bestehen.	
Querbezüge	D-ANF-RV 40	
Kommentare	–	

Titel	*Anforderungsnr.*	*Status*
Protokolle	D-ANF-RV 60	offen
Beschreibung	Sitzungsprotokolle der wöchentlichen Sitzungen sind in das projektinterne SVN hochzuladen, sobald sie in einer weiteren Sitzung abgenommen wurden, d.h. keine Einwände mehr bestehen.	
Querbezüge	D-ANF-RV 40, D-ANF-RV 50	
Kommentare	–	

Kapitel 5
Entwurf

Dieses Kapitel beschreibt den Entwurf unseres Projekts. Dabei richtet sich die Struktur nach einem Bottom-Up Entwurf anhand der Komponenten. Hiervon ausgenommen sind der Zeitgeber zur Systemsynchronisierung der Systemteile aus Abschnitt 5.12 sowie die Konzepte und Aufgaben der Datenbank und des Loggings in Abschnitt 5.13 und Abschnitt 5.14, da diese als Standard betrachtet werden. Das bedeutet, dass dort nicht viel Neuentwicklung stattfindet. Aus diesem Grund wurden die Kapitel als Abschluss an das Entwurfskapitel angehängt.

Für den Entwurf werden zunächst in Abschnitt 5.1 einige Richtlinien zur Modellierung gegeben, nach denen mit Abschnitt 5.2 ein darauf aufbauender Architekturüberblick folgt. Danach beschreibt Abschnitt 5.3 das von uns entwickelte Datenmodell, womit ein Überblick über den Kontext des Systems gegeben ist.
Beginnend mit Abschnitt 5.4 wird nun eine Beschreibung über die verfügbare Hardware gegeben. Hiernach werden in Abschnitt 5.6 die darauf ausgeführten Modelle der Komponenten und mit Abschnitt 5.5 die Komponentenkommunikation beschrieben. Darauf aufbauend erklärt Abschnitt 5.7 das Sampling, womit ein zentraler Baustein unserer Optimierung gegeben ist. Anschließend folgt in Abschnitt 5.8 der Model-Manager zum Initialisieren und setzen von Input-Parametern für Komponenten.
Auf Systemebene erläutert nun Abschnitt 5.9 die Systemkommunikation, welche den Austausch von Nachrichten zwischen den einzelnen Systemen beschreibt. Hiernach werden in Abschnitt 5.10 die Grundlagen der Optimierung aufbauend auf dem Prinzip der Samples entwickelt. Danach wird das Konzept der Benutzeroberfläche in Abschnitt 5.11 beleuchtet. Zuletzt nächstes bietet Abschnitt 5.15 einen Blick auf die Grundlastberechnung nach VDI4655.

5.1 Modellierungsguidelines für Softwaremodellierung

Der Entwurf der Softwarearchitektur ist für die Projektgruppe der Erstellungsprozess einer Grobstruktur des Softwaresystems, welches auf Grundlage der erhobenen Anforderungen im Kern aus einem Energiemanagement- und einem Simulationssystem besteht.

Die Beschreibung der angestrebten Architektur soll Informationen über Struktur und Verhalten des Systems enthalten. Hierfür werden Verteilungssichten und Moduldiagramme benötigt, welche die logische und physische Anordnung der im System enthaltenen Module darstellen. Weiterhin sollen Sequenzdiagramme entworfen werden, damit die Abläufe innerhalb der Teilsysteme Energiemanagement und Simulation sowie die Interaktion untereinander abgebildet werden kann.

Diese Modelle sollen in der Modellierungssprache UML erstellt werden. Zusätzlich dazu soll eine von uns definierte Interface-Modellierung verwendet werden, welche an UML-Klassendiagramme angelehnt ist. Mit diesen sollen neben, einer groben Modulsicht, die nach außen sichtbaren Schnittstellen definiert werden, um das Programmieren in der Gruppe zu erleichtern.

5.2 Architekturüberblick

In diesem Abschnitt wird die Architektur des durch die Projektgruppe zu erstellenden Softwaresystems beschrieben und erläutert. Dazu wird im Sinne einer Top-Down Analyse zuerst die grobe Sicht auf die Systeme beschrieben, dann die Schnittstellen der Systeme angegeben und anschließend Module der Subsystemebene erläutert. Dabei werden, wo angebracht, Verweise auf tiefer gehende Entwurfskapitel geboten.

Zunächst unterteilt sich das Gesamtsystem in drei eigenständige Systeme, das Energiemanagementsystem (EMS), das Simulationsframework (SF) und die Steuerungsebene (SE) .

Das Energiemanagementsystem ist für die Steuerung und Optimierung des Energieverbrauchs zuständig. Das Simulationsframework stellt Daten bereit, die zur Simulation von EVS-Komponenten benötigt werden. Weiterhin wurde in den Systemanforderungen festgehalten, dass EMS und Simulationsframework konfigurierbar sein sollen. Daher wurde ein drittes System, die Steuerungsebene, hinzugefügt, das als Benutzerschnittstelle dient.

Im Rahmen der Projektgruppe soll das EMS als Produkt gesehen werden. Das bedeutet, dass dieses so erstellt werden muss, dass es eigenständig, also ohne die simulierten EVS-Komponenten, funktionieren kann. Infolgedessen wurden EMS und Simulationsframework als eigenständige Programme konzipiert, welche zwar miteinander kommunizieren können, allerdings bis auf kleine Ausnahmen nicht aufeinander angewiesen sind. Abschließend wurde die Steuerungsebene (SE) in diese Zwei-Programm-Lösung integriert, indem sie als Standalone-GUI genutzt wird. (siehe Abbildung 5.1).

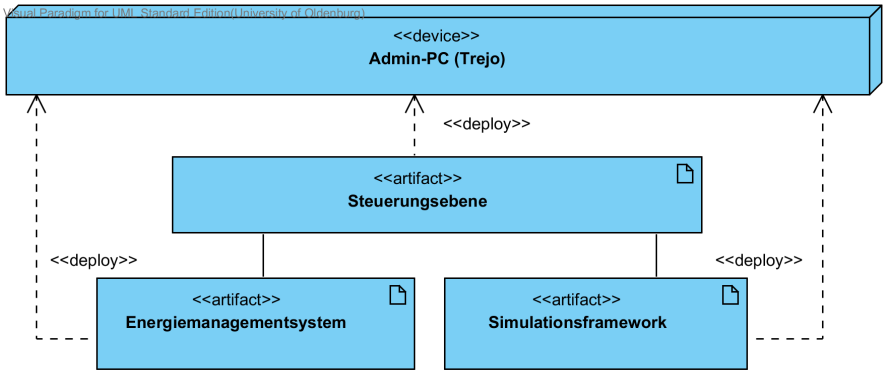


Abb. 5.1 Systemaufteilung in EMS SF und Standalone-GUI zur Steuerung.

Neben diesen drei Systemen existiert eine zentrale Datenbank, welche Konfigurations- und Simulationsdaten speichert, und die simulierten EVS-Komponenten. Erstere wird in Abschnitt 5.13 und letztere in Abschnitt 5.7 näher beschrieben. In diesem Kapitel folgt eine Schnittstellenbeschreibung und eine Beschreibung der drei Systeme.

5.2.1 Schnittstellenbeschreibung

In Abbildung 5.2 sieht man eine Übersicht der Schnittstellen, die im Gesamtsystem verwirklicht werden und über welche die einzelnen Systeme miteinander kommunizieren sollen. In oberster Schicht muss die Steuerungsebene mit dem Energiemanagementsystem und dem Simulationsframework kommunizieren. Dies tut sie über die beiden Schnittstellen EMSCommunication und SIMCommunication. Über diese Schnittstellen werden unter anderem Start, Stoppsignale und Identifikator (ID) ausgewählter Szenarien übermittelt. Nähere Informationen zu diesen Kommunikationsschnittstellen finden sich in Abschnitt 5.9.

Weiterhin sollen die drei Systeme eine Schnittstelle zu einer zentralen Datenbank haben. Das EMS und das SF werden dabei insbesondere den Tagesverlauf der Anlagen und der Optimierung und ihre Fahrpläne in die Datenbank schreiben. Das SE speichert Szenarien und Wetterdaten in die Datenbank, die mittels ID von den beiden anderen Systemen abgerufen werden können.

Eine weitere Schnittstelle besteht zu den EVS-Komponten. Die sogenannte Feldkommunikation hat die Aufgabe Steuersignale (Fahrpläne) an die EVS-Komponenten zu senden oder Werte der EVS-Komponenten dem EMS zu übermitteln. Für simulierte EVS-Komponenten besitzt das Simulationsframework ebenfalls eine Feldkommunikation, über die es EVS-Komponenten mit Simulationsdaten, wie z.B. Wetter oder Sonnenstand, informiert. (Siehe auch Abschnitt 5.5)

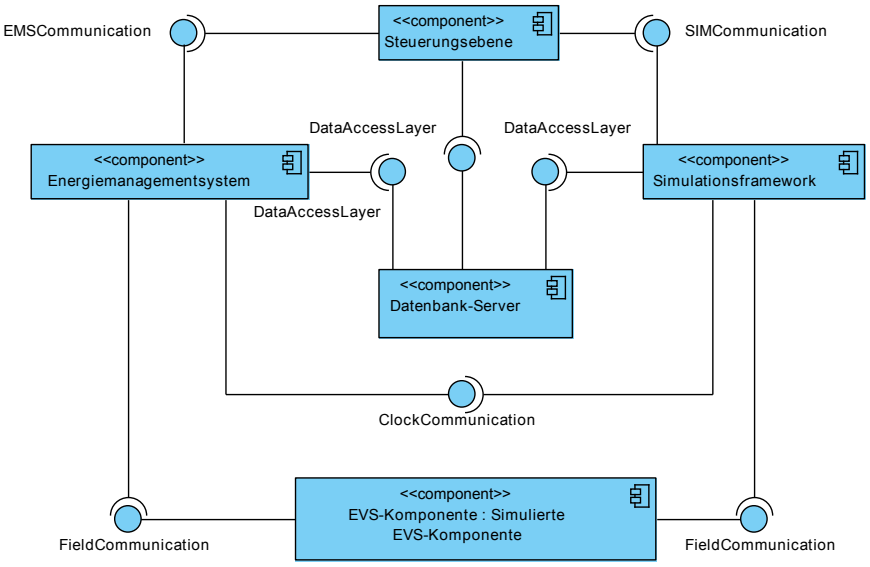


Abb. 5.2 Schnittstellen der drei Systembestandteile

Die letzte Schnittstelle stellt die einzige Verbindung zwischen EMS und SF da. Oben wurde erwähnt, dass die Systeme fast unabhängig sind. Diese Schnittstelle ist dabei die wichtige Ausnahme. Beide Systeme sind zwar prinzipiell unabhängig, aber es ist notwendig, dass ihr Takt synchron bleibt. Denn das SF muss den EVS-Komponenten Simulationsdaten zu dem im EMS aktuellen Simulationsschritt liefern. Um das zu ermöglichen gibt es einen Zeitgeber (Clockgenerator) im EMS, der einen Proxy-Zeitgeber im SF synchron hält. Es wäre aber ohne viel Aufwand möglich diesen durch einen richtigen Zeitgeber zu ersetzen, falls man das Simulationsframework ohne EMS starten möchte. Weitere Informationen zum Zeitgeber finden sich in Abschnitt 5.12.

5.2.2 Energiemanagementsystem

In diesem Abschnitt werden die einzelnen Subsysteme des EMS erläutert. Ein Moduldiagramm mit den Subsystemen und deren wichtigsten Methoden findet sich in Abbildung 5.3

Die Hauptaufgabe des Energiemanagementsystems ist die Optimierung des Eigenverbrauchs. Dazu hat sie die Subkomponente Optimierungssystem, welche die Optimierung der Fahrpläne mithilfe von sogenannten Samples vollzieht. Samples sind mögliche Fahrpläne, welche die EVS-Komponenten dem EMS anbieten. Aus diesen Fahrplänen der einzelnen Komponenten wählt die Optimierung eine Kombination aus, welche die Energiekosten möglichst minimiert. Zur Generierung der

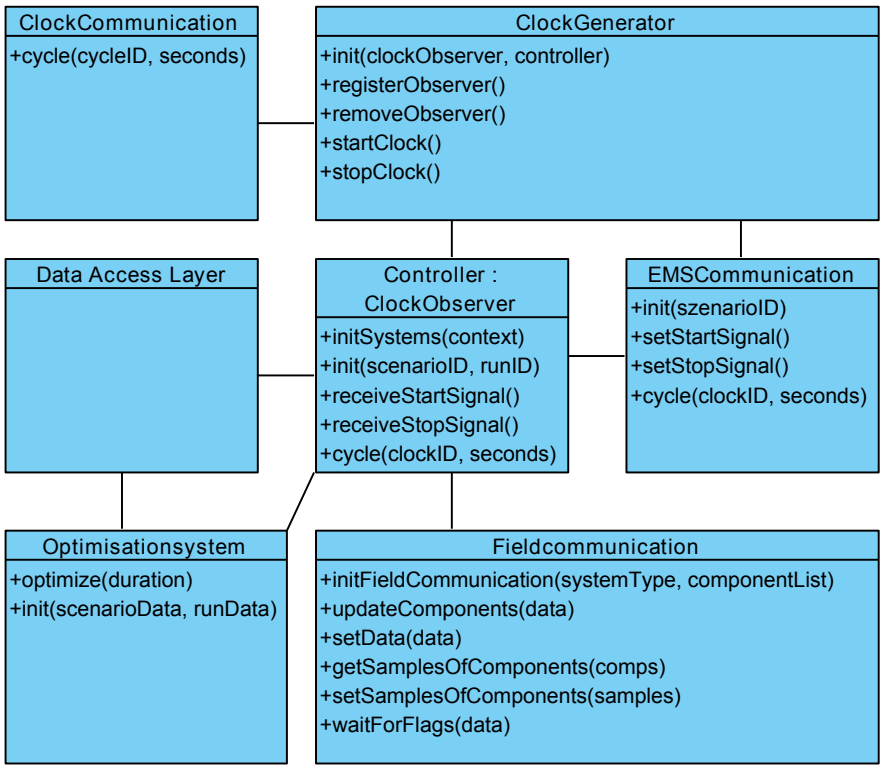


Abb. 5.3 Moduldiagramm des Energiemanagementsystems

Samples finden sich detaillierte Informationen in Abschnitt 5.7 und der Optimierungsalgorithmus wird in Abschnitt 5.10 erläutert.

Die anderen Subsysteme des EMS dienen entweder der Kommunikation mit den anderen Systemen oder regeln den Ablauf des eigenen Systems. Zum einen gibt es den oben erwähnten Zeitgeber, der den Takt des Systems steuert. Sobald das EMS über die EMSCommunication-Schnittstelle ein Startsignal erhalten hat, wird der Zeitgeber gestartet. Dieser regelt den Ablauf der Takte und informiert das SE über die EMSCommunication, das SF über die ClockCommunication und das EMS mittels Controller über fertige Taktzyklen. Der Zeitgeber im EMS ist qausi das Kernstück des Gesamtsystems, von dem jede Berechnung ausgeht.

Der Controller regelt den Ablauf im EMS und sorgt dafür, dass einerseits Daten über die Feldkommunikation abgerufen werden und andererseits dass das Optimierungssystem gestartet wird.

5.2.3 Simulationsframework

Das Simulationsframework ist in Abbildung 5.4 als Moduldiagramm dargestellt.

Im Prinzip ähnelt der Aufbau des SF dem des EMS. Es besitzt, wie das EMS, einen Zeitgeber, einen Controller, eine Kommunikation zur Steuerungsebene und eine Feldkommunikation zu den simulierten EVS-Komponenten.

Anders als im EMS ist der Zeitgeber hier, wie bereits im Schnittstellenabschnitt beschrieben, nur ein Proxy-Zeitgeber. Dieser empfängt über die ClockCommunication Taktsignale und leitet sie an seinem Observer dem Controller weiter. Für diesen ist dabei transparent, dass der Zeitgeber nicht autonom ist, sondern nur Signale weiterleitet. Dadurch ist ein Austausch des Proxy-Zeitgebers durch einen autonomen Zeitgeber ohne Änderung des restlichen Systems möglich.

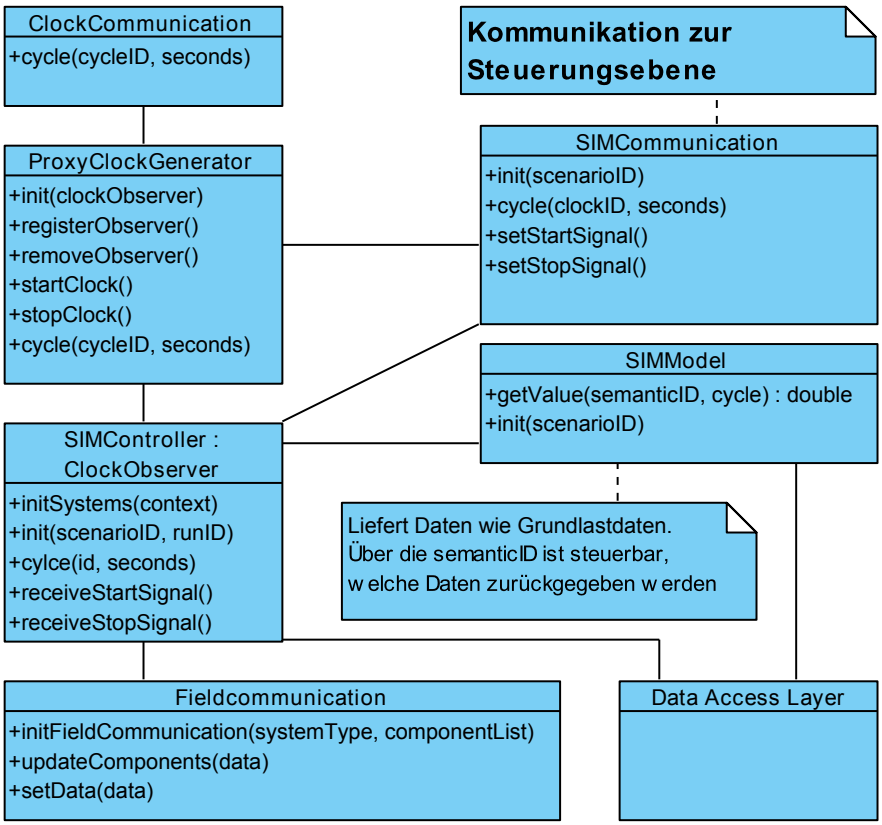


Abb. 5.4 Moduldiagramm des Simulationsframework

Der Controller regelt auch im SF den internen Ablauf des Systems. Dazu ruft es für die simulierten EVS-Komponenten nötige Daten beim SIMModel ab. Dieser

Abruf ist dabei generisch gehalten. In der Datenbank kann das SF abrufen, welche EVS-Komponenten im gewählten Szenario vorliegen und welche Daten diese benötigen. Die Art der Daten ist in einer Semantik-ID verschlüsselt, mit der der Controller ohne Kenntnis der wirklichen Bedeutung die passenden Daten aus dem SIMModel abrufen und mittels Feldkommunikation weiterleiten kann. Das SIMModel hat also die Aufgabe nötige Simulationsdaten zu generieren oder aus der Datenbank abzurufen. Es erfüllt also analog zum Optimierungssystem im EMS die Kernaufgabe des SF. Das SIMModel wird näher in Abschnitt 5.8 behandelt.

5.2.4 Steuerungsebene

Im Folgenden soll die im Moduldiagramm in Abbildung 5.5 dargestellte Steuerungsebene erläutert werden.

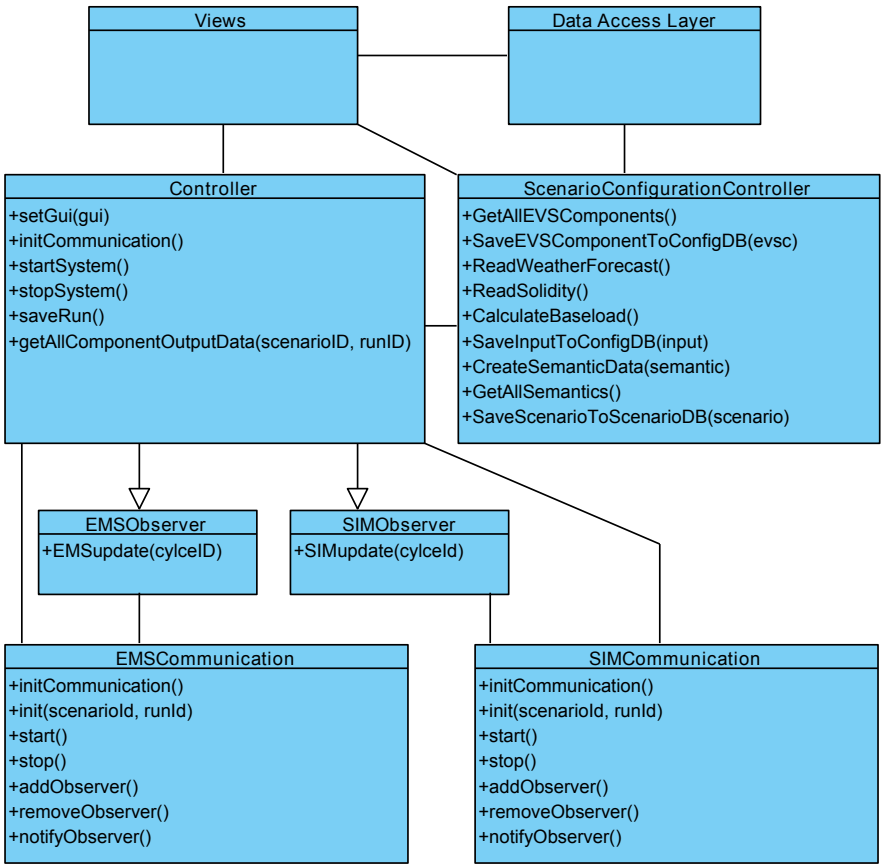


Abb. 5.5 Moduldiagramm der Steuerungsebene

Die Steuerungsebene dient als Benutzerschnittstelle. Hierzu bietet sie dem Nutzer unterschiedliche Views an, in denen dem Benutzer verschiedene Einstellungsmöglichkeiten und Ansichten über Zustände des Energiemanagementsystems, des Simulationsframeworks und der EVS-Komponenten geboten werden. Weiterhin können unter anderem Taktdauer oder der Szenarioaufbau eingestellt werden können. Da die einzelnen Systeme voneinander unabhängig sind, kann die GUI im laufenden Betrieb des EMS ab- oder wieder eingeschaltet werden. Des Weiteren existiert es eine getrennte Evaluationsview, mit der in der Datenbank gespeicherte Laufdaten offline evaluiert und geplotted werden können. Näheres zu den verschiedenen Views findet sich in Abschnitt 5.11.

Neben den Views und den oben beschriebenen Schnittstellen zu den anderen Systemen existieren noch der Controller und der ScenarioConfigurationController. Ersterer ist für die Verwaltung des SE zuständig und behandelt alle Prozesse die im Simulationsbetrieb passieren. Der Controller ist dabei eine Art Zwischenschicht zwischen den Systemen EMS und SF auf der einen Seite und der GUI auf der anderen Seite. Er kann Steuersignale, wie Start und Stoppsignale, an die Systeme senden, oder empfängt als Observer abgeschlossene Zyklen des EMS und SF.

Der ScenarioConfigurationController ist von der Aufgabe des Controller insofern abgerenzt, dass er sich um einmalige Aufgaben kümmert, die nicht während eines Simulationslaufs passieren. So bietet er einerseits Methoden zum Erstellen eines Szenarios oder zur Konfiguration von EVS-Komponenten und andererseits Methoden zum Auslesen von Wetterdaten und Grundlastdaten aus ausgewählten Dateien.

5.3 Datenmodell

Zur Realisierung des Simulations- und zugehörigem Energiemanagementsystem bedarf es zunächst einer Repräsentation des physischen Umfelds, mit der die zu entwickelnde Software Zugriff auf alle notwendigen Informationen erhält. Eine exakte Repräsentation des physischen beziehungsweise realen Umfelds ist dabei nicht praktikabel, da diese deutlich zu viele Daten enthalten müsste, die auch nur teilweise für die Simulation berücksichtigt werden müssten. Aus diesem Grund wurde sich für eine abstrakte Darstellung der Umwelt entschieden. Um eine passende Abstraktion finden zu können, wurden zunächst die Daten gesammelt, die grundlegenden Einfluss auf das System haben und somit relevant für die Simulation sind.

Datensammlung

Mit der Datensammlung sollen die Daten des physischen Umfelds auf für die Simulation relevante Daten gefiltert werden. Das Kernelement der Simulation bildet das Gebäude, welches durch eine prognostizierte Energiegrundlast gekennzeichnet ist und wiederum aus verschiedenen EVS-Komponenten besteht. Zu den EVS-Komponenten des Gebäudes zählen Energieerzeuger, -speicher und -verbraucher. Energieerzeuger sind Photovoltaik-Anlagen (elektrische Energie) und Blockheizkraftwerke (elektrische und thermische Energie). Energiespeicher stehen in Form von Batteriespeichern für elektrische Energie und Pufferspeichern für thermische Energie zur Verfügung. Des Weiteren gibt es steuerbare Verbraucher, die zusätzlich zur Grundlast des Gebäudes Energie verbrauchen. Die prognostizierte Grundlast wird, um die Simulation realistisch zu gestalten, vom Simulationsframework (SF) zu einer tatsächlichen Grundlast verrauscht. Das Energiemanagementsystem (EMS) kennt lediglich die Prognose und muss auf die Variabilität dieser Daten reagieren.

Das Ziel des EMS ist die Eigenverbrauchsoptimierung des gesamten Gebäudes zur Minimierung der Energiebezugskosten im Rahmen der Simulationsdauer. Aus dieser Zielsetzung ergibt sich, dass das EMS das Verhalten der EVS-Komponenten kennen muss und je nach Art der Komponente das Verhalten steuern kann. Als weiterer Umgebungsparameter wurde das Energieversorgungsunternehmen identifiziert, welches das Gebäude mit Energie versorgen kann beziehungsweise der Einspeisung überschüssiger Energie dient. Das Energieversorgungsunternehmen muss durch Preise für den Energiebezug und Vergütung für die Einspeisung gekennzeichnet sein, damit das EMS bestimmte Entscheidungen treffen kann. Beispielhafte Fragestellungen für das EMS lauten: Ist es zum aktuellen Zeitpunkt günstiger das Blockheizkraftwerk laufen zu lassen um den Bedarf an elektrischer Energie zu decken als den Strom vom Energieversorgungsunternehmen zu beziehen? Erzeugt die PV-Anlage ausreichend elektrische Energie zur Versorgung des Gebäudes und kann das BHKW abgeschaltet werden? Lohnt sich die Einspeisung von elektrischer Energie zum aktuellen Zeitpunkt?

Als weiterer Umgebungsparameter wurde der Faktor Wetter und Klima identifiziert. Dieser Faktor hat ebenfalls großen Einfluss auf das Simulationssystem, da

beispielsweise geringere Temperaturen zu erhöhten Bedarf an thermischer Energie führen oder die Strahlung Einfluss auf die Produktion von elektrischer Energie der PV-Anlage hat. Die Wetterdaten müssen zum Einen in Form von Prognosedaten zur Verfügung stehen, damit das EMS Fahrpläne für die EVS-Komponenten erstellen kann, zum Anderen zur realitätsnahen Repräsentation der Umwelt vom Simulationsframework verrauscht werden. Auf Abweichungen der verrauschten im Vergleich zu den prognostizierten Werten muss das EMS reagieren.

In Abbildung 5.6 sind die auftretenden Komponenten und deren Beziehungen zueinander für ein beispielhaftes Szenario, welches den Anforderungen an das System entspricht, dargestellt. Eine Möglichkeit zur Realisierung der Software wäre die Reduzierung des Datenmodells auf die herausgestellten Daten und Parameter, was jedoch dazu führen würde, dass dem späteren Anwender der Software nur eine geringe Anzahl an Konfigurationsmöglichkeiten für ein Szenario zur Verfügung stünden. Daher wurde sich im Sinne der Erweiterbarkeit des Systems für die Entwicklung eines abstrakten Datenmodells entschieden.

Abstraktion

Bereits beim Sammeln der Informationen, die das Szenario/Umfeld darstellen, zeigt sich deutlich, dass Abhängigkeiten zwischen den vorhandenen EVS-Komponenten und den erforderlichen Umweltdaten bestehen. So benötigt beispielsweise eine PV-Anlage lediglich Wetterinformationen, die die Sonneneinstrahlung betreffen.

Angenommen, die Simulation erfolgt für ein festes Szenario, so könnte im Datenmodell die Umwelt auf die benötigten Daten reduziert werden. Gegensätzlich zu dieser Annahme soll die Simulation mit variablen Szenarien erfolgen. Ein festes Datenmodell würde somit aufgrund der Abhängigkeiten der EVS-Komponenten zu den benötigten Daten das entstehende System in seiner Erweiterbarkeit stark einschränken, da alle zum Zeitpunkt der Entwicklung nicht berücksichtigten Komponenten nur im Ausnahmefall von dem Model abgedeckt werden. Um eine solche Einschränkung umgehen zu können, wird eine Abstraktion von einem direktem Datenmodell auf ein Datenmodell benötigt, mit dem nicht nur die Werte der Datensätze sondern auch der Aufbau dieser Datensätze selbst variabel beschrieben werden kann.

Diese Abstraktion führt dazu, dass es in dem Datenmodell nicht mehr klar Wetterdaten, Grundlastdaten, etc. gibt, sondern Inputdaten, denen beliebig viele Inputattribute zugeordnet sind. Den Attributen sind dabei wiederum beliebig viele Inputwerte zugeordnet. Mit diesem Modell der Inputdaten können die Datensätze beliebig erweitert und auch neue Datensätze, die bisher nicht vorgesehen waren, hinzugefügt werden ohne eine Änderung am Modell vornehmen zu müssen.

So wie für die Inputdaten wird eine Abstraktion für die EVS-Komponenten benötigt, mit der alle Komponenten in gleicher Form abgebildet werden können. Die vorgenommene Abstraktion kennt dabei nicht mehr direkt PV-Anlagen oder Blockheizkraftwerke, sondern nur noch allgemein EVS-Komponenten. Den EVS-Komponenten werden dabei, so wie auch den Inputdaten, beliebig viele Attribute

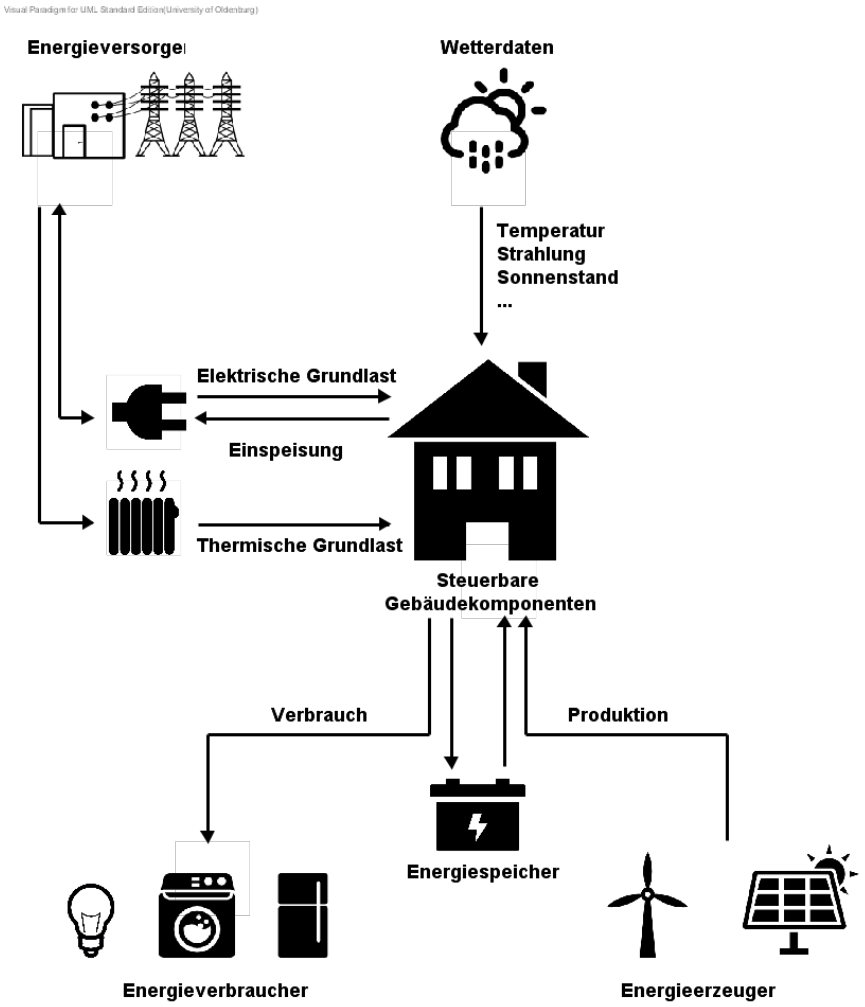


Abb. 5.6 Überblick der Daten

zugordnet, die die Parameter der Komponenten beschreiben. Dabei ist es unerheblich, ob es sich um Steuerparameter oder Simulationsparameter (z.B. für Wetterdaten) handelt. Diese Unterscheidung wird erst durch die Beschreibung des Attributs selbst wieder deutlich.

Diese Abstraktionen lösen zwar das Problem, beliebig erweiterbare Szenarien darzustellen, führt aber auch zu dem Problem, dass die Daten nicht mehr eindeutig zugeordnet werden können. So wäre es bei einem festen Modell klar gewesen, dass zum Beispiel der Input-Parameter „Sonnenstand“ einer PV-Anlage fest mit selbigem Wert aus den Wetterdaten beschrieben werden muss. Dieses Mapping der Parameter

auf Inputdaten muss im abstrakten Modell auch wieder ermöglicht werden. Hierzu wurden sowohl die Inputattribute der Inputdaten als auch die Attribute der EVS-Komponenten mit Semantiken ausgestattet. Stimmt die Semantik eines Attributs mit der eines Inputattributs überein, wird es mit diesem beschrieben. Dabei ist jedoch zu beachten, dass die bei den Inputdaten vorhandenen Semantiken einzigartig sind, da sonst kein deterministisches Mapping möglich ist.

Mit diesem abstrahierten Datenmodell ist es nun möglich das System mit allen wichtigen Informationen zu versorgen, die für die Simulation benötigt werden.

5.4 Hardwareaufbau

Als Hardware für die EVS-Komponenten stehen drei Embedded PC's der Firma Beckhoff zur Verfügung. Die PC's vom Typ CX2020 sind jeweils mit einem 1,4 GHz Intel Celeron Prozessor und 2 GB Arbeitsspeicher ausgestattet. Untereinander sind die PC's per Ethernet und EtherCAT, einem Realzeit-Ethernet Protokoll, verbunden. Jeder der PC's verfügt über die TwinCAT-Runtime, auf der Software-Module in Echtzeit ausgeführt werden können. Für die Kommunikation mit den EVS-Komponenten steht ein OPC Unified Architecture (OPC-UA) Server zur Verfügung, der speziell die Ansteuerung der Komponenten der TwinCAT-Runtime ermöglicht.

Abb. 5.7 Hardwareaufbau im PG-Raum

Die in Abschnitt 5.2 beschriebenen Systeme werden auf einem von den Embedded PC's separatem Rechner ausgeführt und nur die EVS-Komponenten des zu simu-

lierenden Gebäudes werden als solche Module auf den PCs ausgeführt. Die Software verwendet diese simulierten Komponenten um das Verhalten realer Komponenten für die Optimierung nachempfinden zu können.

5.5 Komponentenkommunikation

Für die Simulation und Optimierung von EVS-Komponenten wird eine Schnittstelle für die Komponentenkommunikation benötigt.

Auswahl des Protokolls

Neben den in Abschnitt Abschnitt 4.3 beschriebenen Anforderungen an die Funktionalität der Komponentenkommunikation bestehen bereits Anforderungen an die Ansteuerung beziehungsweise das Kommunikationsprotokoll der EVS-Komponenten.

- Ansteuerung von simulierten und realen EVS-Komponenten
- Protokoll soll ansteuerbare EVS-Komponenten-Typen nicht einschränken
- Protokoll soll möglichst verbreitet sein

Das erste Protokoll, dass zur Erwägung gezogen wurde ist *EtherCAT*. EtherCAT ist eine Realzeit-Ethernet Lösung von der Firma Beckhoff. Mit diesem Protokoll können Daten sehr zuverlässig und schnell (innerhalb von 30-100 μs) ausgetauscht werden. Die Konfiguration dieses Protokolls stellte sich jedoch als relativ komplex heraus, so dass wir andere Protokolle bevorzugt haben.

Das zweite Protokoll ist Automation Device Specification (ADS). ADS ist eine Geräte- und Feldbusunabhängige Schnittstelle, mit der ADS Teilnehmer kommunizieren können. Vorteil dieses Protokolls ist, dass die Hardware, auf der die Komponenten ausgeführt werden, dieses Protokoll unterstützt. Weiterhin ist ADS in seiner grundsätzlichen Handhabung ähnlich zu POSIX-Sockets, mit denen sehr leicht Bytes übertragen werden können. Nachteilhaft ist, dass unklar ist, ob dieses Protokoll für reale Komponenten verwendet wird und wie verbreitet es ist.

Das dritte Protokoll ist *IEC 61850*. Dieses standardisierte Übertragungsprotokoll wird für die Kommunikation mit dezentralen Energieerzeugungsanlagen verwendet. Die Verwendung dieses Protokolls hätte somit den Vorteil, dass die Kompatibilität zu realen Komponenten direkt gegeben ist. Der Nachteil dieses Protokolls besteht jedoch in der Komplexität und dem Aufwand, das Protokoll in die Software zu integrieren, insbesondere da der eigentliche Standard uns nicht vorliegt.

Als viertes Protokoll wurde *OPC-UA* untersucht. OPC-UA ist ein standardisiertes (siehe IEC 62541) und industrielles Kommunikationsprotokoll, dass von der OPC Foundation entwickelt wurde. So wie auch bei IEC 61850 lässt dies darauf schließen, dass es sich bei OPC-UA um ein in der Industrie verbreitetes Protokoll handelt. Für dieses Protokoll stehen auch bereits verschiedene Implementationen bereit, so dass eine Einbindung von OPC-UA für die Komponentenkommunikation leicht rea-

lisierbar sein sollte. Auch für die Industrie-PC's, auf denen die EVS-Komponenten ausgeführt werden, existiert bereits eine UPC-UA-Server Implementierung von der Firma Beckhoff, die die Schnittstellen der TwinCAT-Module um OPC-UA erweitert.

Von den untersuchten Protokollen eignet sich OPC-UA am besten für die Realisierung der Komponentenkommunikation, da es standardisiert ist - so wie auch IEC 61850. Im Gegensatz zu IEC 61850 sollte der Implementationsaufwand aufgrund der vorhandenen Client-Bibliothek und auch einer serverseitigen Implementation relativ gering sein.

Entwurf der Komponentenkommunikation

Für die Modellierung der Komponentenkommunikation gab es generell zwei Ansätze. Der eine Ansatz sieht eine direkte Verwendung der Komponenten über OPC-UA vor. Hier wird die Verwaltung und Kontrolle der EVS-Komponenten den zugreifenden Systemkomponenten überlassen. Der andere Ansatz schaltet eine zentrale Steuerungskomponente (weiterhin *EVS-Component-Manager* genannt) zwischen die System- und die EVS-Komponenten. Durch die Zwischenschaltung des EVS-Component-Managers kann dieser Zugriffe auf mehrere EVS-Komponenten optimierter ausführen (zum Beispiel durch parallele Ausführung dieser Zugriffe). Die Verwendung eines EVS-Component-Managers führt auch zu einer leichteren Handhabung der Komponenten bei Ausfällen oder beim Starten anderer Simulationen, da nur noch der EVS-Component-Manager über solche informiert werden muss und dieser dann die betroffenen EVS-Komponenten korrekt beenden kann. Aufgrund dieser Vorteile basiert die entworfene Komponentenkommunikation auf der indirekten Variante über einen EVS-Component-Manager.

Um eine Einschränkung des Systems auf OPC-UA zu vermeiden, wurde bei der Modellierung der eigentlichen EVS-Komponenten eine Schnittstelle für solche entworfen. Mit dieser Schnittstelle können dann EVS-Komponenten erstellt werden, die verschiedene Kommunikationsprotokolle verwenden. Durch diese Schnittstelle wird offen gehalten noch weitere Protokolle zusätzlich zu OPC-UA umzusetzen.

Die in Anforderung EMS-80A, EMS-100A und SF-90 geforderten Funktionen sind dabei in der Schnittstelle der EVS-Komponente und des EVS-Component-Managers berücksichtigt und entsprechend in der späteren Umsetzung dieser Struktur realisiert.

5.6 Komponentenmodelle

Wie im Ergebnis der Problemanalyse (siehe Abschnitt 3.3) festgehalten, sollen in der Simulation verschiedene Komponentenmodelle zum Einsatz kommen. Es sind drei Komponentenmodelle für die Erzeugung oder Speicherung thermischer bzw. elektrischer Energie und zwei für den Verbrauch dieser Energie vorgesehen. Als Erzeuger dienen ein Blockheizkraftwerk (BHKW) und eine Photovoltaik-Anlage (PV-

Anlage). Für die Speicherung kommt ein vereinfachtes Modell einer Batterie zum Einsatz. Der Verbrauch wird durch eine abzuschätzende, nicht-steuerbare Grundlast beschrieben und durch zwei steuerbare Verbraucher, die im folgenden noch genauer spezifiziert werden.

Für das Blockheizkraftwerk sowie für die Photovoltaikanlage wurden uns in Matlab-Simulink geschriebene Modelle zur Verfügung gestellt. Für die Batterie liegt uns ein vereinfachtes Modell in textueller Form vor. Für die Grundlast und die steuerbaren Verbraucher liegen keine Modelle vor. Sie müssen eigenständig entworfen werden.

Alle Komponentenmodelle müssen so entwickelt oder angepasst werden, dass sie auf einem Koppler des Unternehmens Beckhoff ausführbar sind. Für die Prototypen muss es möglich sein, über die Schnittstelle ADS zu kommunizieren. Später wird über das Protokoll OPC-UA kommuniziert (vgl. Abschnitt 5.5). Dafür müssen die Modelle in sogenannte TwinCAT Module überführt werden. Ein TwinCAT Modul besteht aus einer Menge an formell definierten Eigenschaften, die den allgemeinen Umgang mit diesem Modul beschreiben. Solch ein Modul kann in der TwinCAT Runtime instanziiert werden und auf einem Beckhoff-Koppler zum Laufen gebracht werden. Jedes Modul enthält eine sogenannte ITComObject-Schnittstelle. Über diese ist es möglich, Parameter des Moduls oder spezielle ADS-Ports anzusprechen. Die ADS-Ports werden vom Entwickler implementiert und können dazu dienen, Eingabe- und Ausgabewerte zu übermitteln.

Damit verschiedene Komponentenmodelle die Simulation energieautonomer Gebäude ermöglichen, ist eine Steuerung der Zusammenarbeit notwendig. Dafür gibt es ein Optimierungsverfahren, welches auf ein Sampling der Komponentenmodelle aufsetzt. Dieses wiederum bedingt aber, dass der interne Zustand eines Komponentenmodells auslesbar ist.

5.6.1 BHKW

Das als Matlab-Simulink Modell gegebene BHKW stellt eine nicht zuletzt für die Optimierung besondere Komponente dar. Das BHKW ermöglicht nicht nur die Erzeugung von elektrischer und thermischer Energie, sondern stellt auch noch einen Warmwasserspeicher bereit. Es kann anhand zahlreicher Parameter spezifiziert werden. Die Tabelle 5.8 gibt einen vollständigen Überblick über alle Eingaben des BHKWs.

Damit das BHKW auf einem Beckhoff-Koppler ausgeführt werden kann, muss aus dem Matlab-Simulink Modell ein TwinCAT Modul generiert werden. Beckhoff bietet dafür das Produkt TE1400 (TwinCAT Target für Matlab-Simulink). Zusammen mit dem Simulink-Coder der Matlab-Simulink Umgebung generiert das TwinCAT Target für Matlab-Simulink bei entsprechender Konfiguration ein TwinCAT Component Object Model (TcCom). Ein TcCom hat dasselbe Ein- und Ausgangsverhalten wie das ihm zugrunde liegende Matlab-Simulink Modell. Das TcCom definiert dabei die Charakteristiken und das Verhalten eines TwinCAT-Moduls. Es kann direkt in

Parameter	Einheit	Anmerkung
CHP.cosPhi	1	Verhältnis Wirk- zu Blindleistung
CHP.chpCoefficient	1	Stromkennzahl
CHP.maxThermalPower	W	maximale thermische Leistung
CHP.maxActivePower	W	maximale elektrische Leistung
CHP.minStartTemp	°C	minimale Rücklauftemperatur
CHP.startingPowerPct	1	prozentuale Anlagenleistung beim Anfahren
CHP.powerLosses	1	prozentuale Leistungsverluste
CHP.maxSupplyTemp	°C	maximale Vorlauftemperatur
Kontrollinformation	**Einheit**	**Anmerkung**
CHP.nominalTemp	°C	Nominaltemperatur Rücklauf für Anlagenregelung
CHP.nominalTempDiff	K	Regelungsband der Nominaltemperatur
CHP.schedule	W	Anlagenfahrplan (Vorgabe: elektrische Leistung in W)
CHP.mode	-	Betriebsmodus des BHKW (Modus 0: fahrplanorientiert, Modus 1: 2-Punkt-Regelung, Modus 2: Drehzahlmodulation, Modus 3: Kombination aus 1 und 2, Modus 4: wärmegeführt, Modus 5: elektrischgeführt)
Input	**Einheit**	**Anmerkung**
Building.thermalLoad	kWh	thermischer Enegiebedarf des Gebäudes (ACHTUNG: kWh)
Building.electricalLoad	kW	elektrische Last des Gebäudes
Parameter	**Einheit**	**Anmerkung**
BWS.volume	m^3	Speichervolumen
BWS.height	m	Speicherhöhe
BWS.numberOfLayers	1	Anzahl der simulierten Wasserschichten
BWS.lossCoefficient	?	Wärmeverlustkoeffizient
BWS.totalLossCoefficient	*1*	*wird nicht benutzt*
BWS.heatTransferCoefficient	?	Wärmetransport zwischen Wasserschichten des Speichers
BWS.boilingPoint	°C	Siedetemperatur Wasser
Kontrollinformation	**Einheit**	**Anmerkung**
BWS.heaterMode	*1*	*sollte nicht benutzt werden*
BWS.nodeHeater1	*1*	*sollte nicht benutzt werden*
BWS.nodeThermostat1	*1*	*sollte nicht benutzt werden*
BWS.setPointTemperature1	*°C*	*sollte nicht benutzt werden*
BWS.deadband1	*K*	*sollte nicht benutzt werden*
BWS.maxHeatingRate1	*W*	*sollte nicht benutzt werden*
BWS.nodeHeater2	*1*	*sollte nicht benutzt werden*
BWS.nodeThermostat2	*1*	*sollte nicht benutzt werden*
BWS.setPointTemperature2	*°C*	*sollte nicht benutzt werden*
BWS.deadband2	*K*	*sollte nicht benutzt werden*
BWS.maxHeatingRate2	*W*	*sollte nicht benutzt werden*
Input	**Einheit**	**Anmerkung**
BWS.ambientTemperature	°C	Umgebungstemperatur

Abb. 5.8 Eingabeparamter des BHKWs

der TwinCAT-Entwicklungsumgebung instanziiert werden. Da ein TcCom statisch ist, kann es nicht versehentlich umprogrammiert werden.

Um das Matlab-Simulink Modell des BHKW mit Hilfe des TwinCAT Target für Matlab-Simulink in ein TcCom zu überführen, muss das Matlab-Simulink Modell zur Codegenerierung geeignet sein. Andernfalls schlägt die Codegenerierung des

Simulink-Coders fehl, auf die das TwinCAT Target für Matlab-Simulink aufsetzt und es kann kein TcCom erstellt werden. Im verwendeten Matlab-Simulink Modell des BHKWs gibt es allerdings drei sogenannte S-Function-Blöcke. Diese werden genauer auch als M-S-Function-Blöcke bezeichnet, da sie in der Sprache Matlab (.m) geschrieben sind. Ein S-Function-Block erlaubt es, ein Matlab-Simulink Modell um benutzerdefinierte Funktionalität zu ergänzen. Dabei kann das Modell auf einfache Weise auch um Zustände ergänzt werden, die diskrete oder kontinuierliche Dynamiken erlauben. Dies allerdings sorgt dafür, dass aus einem Matlab-Simulink Modell, das S-Functions verwendet, nicht direkt Code generiert werden kann. Bei der Implementierung des BHKWs muss dieses Problem gelöst werden.

5.6.2 PV-Anlage

Die PV-Anlage ist ebenfalls als Matlab-Simulink Modell gegebenen. Die Tabelle 5.9 gibt einen Überblick über alle Eingaben und Ausgaben der PV-Anlage.

Parameter	Einheit	Beschreibung
PV.NOCT	°C	NOCT-Wert
PV.ID	keine	Albedo
PV.Inverter.maxActivePower	W	Maximalleistung Wechselrichter
PV.Angle	°	Neigungswinkel
PV.NumberOfModulesPerString	keine	Modulanzahl pro String
PV.NumberOfStrings	keine	Stringanzahl
PV.IMPP	A	Nennstrom
PV.aP	%/K	Temperaturkoeffizient Leistung
PV.UMPP	V	Nennspannung
Input	**Einheit**	**Beschreibung**
Environment.sunRadiation	W/m²	Globalstrahlung
Environment.sunElevation	°	Sonnenhöhe
Environment.sunAzimuth	°	Sonnenazimut
Environment.temperature	°C	Außentemperatur
Environment.Albedo	keine	Umgebungsalbedo
Output	**Einheit**	**Beschreibung**
PV.electricalPower	W	Elektrische Leistung

Abb. 5.9 Ein- und Ausgaben der PV-Anlage

Auch das Modell der PV-Anlage soll mit Hilfe des TE1400 in ein TcCom überführt werden. Allerdings enthält es analog zum BHKW drei M-S-Function-Blöcke, die nicht zur Codegenerierung geeignet sind. Es besteht also das gleiche Problem.

5.6.3 Batterie

Bestandteil des Systems zur Simulation eines Gebäudes ist eine Batterie als Speicher, wie in Abschnitt 3.1 beschrieben. Grundlage für den Entwurf der Batterie ist ein in Formeln vorliegendes Batteriemodell der Betreuer gewesen. Dieses Modell vereinfacht eine Batterie sehr stark, indem Außentemperatur, Temperatur der Batterie, Wirkungsgrade sowie die Alterung der Batterie vernachlässigt werden. Die zugrundeliegenden Formeln des Modells finden sich in den Gleichungen 5.1 bis 5.8.

Je nach chemischer Zusammensetzung einer Batterie sind verschiedene Ladeverfahren notwendig. Für das Batteriemodell in diesem Projekt wird ein CCCV Ladeverfahren verwendet. Der Ladestand einer Batterie während dieses Ladeverfahrens wird in Abbildung 5.10 visualisiert. CCCV Ladeverfahren bedeutet, dass zunächst mit einem konstanten Strom (constant current, cc) geladen wird. Dies ist in der oberen Teilgleichung von Gleichung 5.8 modelliert und in Abbildung 5.10 bis ca Minute 120 zu sehen. Durch das Laden mit konstantem Strom wird ein zu hoher Ladestrom vermieden. Ab dem Schwellenwert t_{cc} (s. Gleichung 5.2) wird mit einer konstanten Spannung geladen, um zu verhindern, dass die Ladeschlussspannung überschritten wird. Als Ladeschlussspsannung werden im Modell 99% angenommen, wie in der unteren Teilgleichung von Gleichung 5.7 zu sehen ist. Wie sich dies auf die Eingangsleistung auswirkt, kann in Abbildung 5.10 ab Minute 120 betrachtet werden.[36]

Wie im Abschnitt 2.3.3 des Seminarbands beschrieben, werden als Eingangsparameter die Eingangsleistung P_{crg}, der Wirkungsgrad und der Energieverlust benötigt.

Neben diesen im Seminarband bereits herausgearbeitet Werten wird als weitere Parameter eingegeben, wie groß die maximale Kapazität E_{max} ist und wie lang ein Simulationsschritt sein soll. Kleinere Zeitschritte ermöglichen eine genauere Berechnung, insbesondere wenn in dem Zeitschritt die Ladestrategie gewechselt werden muss. Andererseits benötigen feinere Zeitschritte auch mehr Berechnungen. In den Formeln ist dieser Wert als Δt zu finden. Zusätzlich kann auch angegeben werden, wie lang ein Lade- oder Entladezyklus mindestens sein soll. Dieser gibt in Minuten an, wie oft gewechselt werden kann. Bei einer Eingabe von 30 Minuten darf vom einem Ladezyklus zu einem Entladezyklus oder umgekehrt frühestens nach 30 Minuten gewechselt werden. Dabei zählen Phasen, in denen weder geladen noch entladen wird, zu der Phase, in welcher die Batterie zuvor gewesen ist.

Der Energieverlust hingegen wird nicht berücksichtigt, wie bereits oben bei den Vereinfachungen erläutert.

Typische Werte für die Ladeleistung P_{crg} einer Batterie finden sich in Gleichung 5.1.

$$\begin{aligned} Normalstorm(1-Phasen-Wechselstrom) &: P_{crg} = 3,7kW \\ Starkstrom(3-Phasen-Wechselstrom) &: P_{crg} = 11kW \end{aligned} \tag{5.1}$$

In diesem Modell wird die Zeitspanne des Konstantspannungs - Ladeverfahrens angegeben, wie in 5.3 definiert, um daraus den Zeitpunkt des Wechsels vom Konstant-

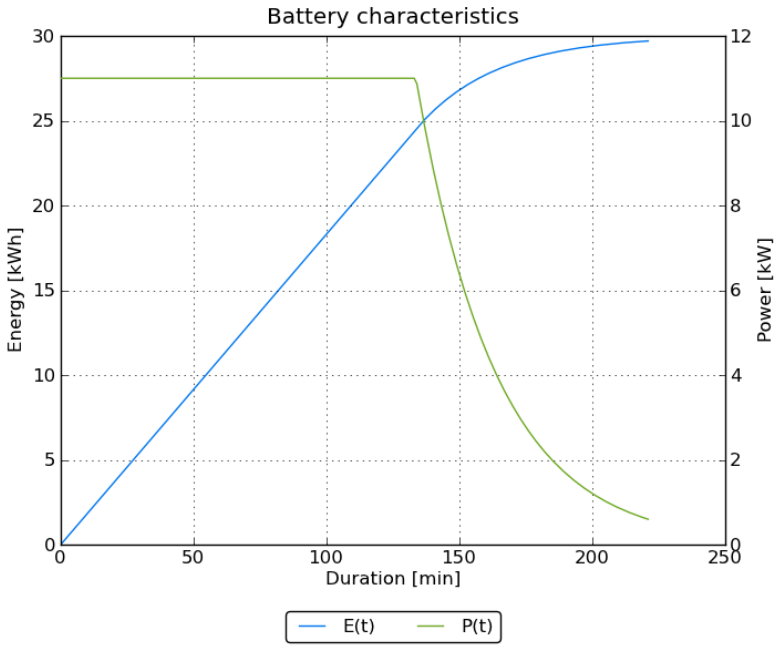


Abb. 5.10 Ladestand und Eingangsleistung einer Batterie mit CCCV Ladeverfahren

strom - Konstandspannungs - Ladeverfahrens zu berechnen. Der Zeitpunkt ergibt sich aus dem Verhältnis der mit konstantem Strom aufgenommenen Energiemenge zu der Ladeleistung, wie in Gleichung 5.2 zu sehen ist. Die mit konstantem Strom aufgenommene Energiemenge ergibt sich widerum aus der Kapazität der Batterie und der mit konstanter Spannung geladenen Energiemenge, wie in Gleichung 5.4 formalisiert wurde. Die mit konstanter Spannung geladene Energiemenge ergibt sich aus dem Anfangs erläuterten Zeitspanne des Konstantspannungs - Ladeverfahren abhängig von der Ladeleistung, wie in Gleichung 5.5 zu sehen. Aus dieser Formel ergibt sich, dass die als Zeitspanne des Konstantspannungs - Ladeverfahren angegebene Zeitdauer nicht der Zeitdauer entspricht, die das Konstantspannungs - Ladeverfahren benötigt. Denn in der Gleichung 5.5 wird die Energiemenge des Konstantspannungs - Ladeverfahrens mit der Zeitdauer und der maximalen Ladeleistung bestimmt. Allerdings lädt das Konstantspannungs - Ladeverfahren nur zu Beginn mit der maximalen Ladeleistung und danach mit einer immer gerinergen Leistung, wie in Gleichung 5.6 zu sehen ist.

$$t_{cc} = \frac{E_{cc} \cdot \Delta t}{P_{crg}} \tag{5.2}$$

k := Zeitspanne des Konstantspannungs - Ladeverfahren in Sekunden (5.3)

$$E_{cc} = E_{max} - E_{cv} \tag{5.4}$$

$$E_{cv} = \frac{P_{crg} \cdot k}{\Delta t} \tag{5.5}$$

In Gleichung 5.6 wird die Leistungskurve der Batterie dargestellt. Die erste Teilfunktion modelliert das konstante Ladeverfahren, bei dem die Ladeleistung maximal ausgenutzt werden kann. In der zweiten Teilfunktion wird die Leistungskurve bei konstanter Spannung dargestellt. Hier wird die Ladeleistung abhängig vom Ladestand der Batterie begrenzt, um ein Überladen der Batterie zu verhindern.

$$P(t) = \begin{cases} P_{crg}, & \text{wenn } 0 \leq t < t_{cc} \\ min(P_{crg}, E_{cv} \cdot exp(\frac{-(t-t_{cc})}{k}) \cdot \frac{\Delta t}{k}), & \text{wenn } t \geq t_{cc} \wedge E(t) \leq 0,99 E_{max} \end{cases} \tag{5.6}$$

Die Gleichung 5.7 für die Energiefunktion, solange noch nicht der Übergang zum Konstantspannungs - Ladeverfahren erreicht wurde, besteht aus der Aufnahmeleistung der Batterie P_{crg} multipliziert mit der vergangenen Zeit und anschließend dividiert durch die gewählte Zeitintervallgröße. Die Energie nach dem Übergang zum Konstantspannungs - Ladeverfahren ist abhängig von der noch möglichen Ladeleistung, die in Gleichung 5.6 definiert wurde.

$$E(t) = \begin{cases} \frac{P_{crg}}{\Delta t} \cdot t, & \text{wenn } 0 \leq t < t_{cc} \\ E_{cc} + E_{cv} \cdot (1 - exp(-\frac{t-t_{cc}}{k}), & \text{wenn } t \geq t_{cc} \wedge E(t) \leq 0,99 \cdot E_{max} \end{cases} \tag{5.7}$$

Abhängig vom derzeitigen Ladestand der Batterie kann der Zeitpunkt errechnet werden, in dem sich die Batterie gerade befindet. Die Gleichungen dafür finden sich in Gleichung 5.8. Auch bei der Berechnung des Zeitpunktes muss zwischen Konstantstromverfahren in der ersten Teilgleichung und dem Konstantspannungsladeverfahren in der zweiten Teilgleichung unterschieden werden.

$$t(E) = \begin{cases} E/m, & \text{wenn } 0 \leq t < t_{cc} \\ t_{cc} - k \cdot log(1 - \frac{E-E_{cc}}{E_{cv}}), & \text{wenn } t \geq t_{cc} \wedge E(t) \leq 0,99 E_{max} \end{cases} \tag{5.8}$$

Die hier vorgestellten Gleichungen sowie die erläuterten zusätzlichen Parameter sind die Grundlage für die Implementierung des Batteriemodells in Abschnitt 6.8.2.

5.6.4 Steuerbarer Verbraucher

Als erster steuerbarer Verbraucher wird zunächst ein einfaches Modell zum Einsatz kommen, das eine Waschmaschine präsentiert. Dieses Modell gibt als einzige Ausgabe den Stromverbrauch an, den die Waschmaschine hat. Die Einheit des Ausgabewerts ist Kilowatt. Die Ausgabe bezieht sich jeweils auf das folgende 15-minütige Zeitintervall. Als Eingabewerte erwartet das Modell zwei Parameter, die die Beladung in Kilogramm und die Temperatur in Grad Celsius angeben sowie einen Parameter, der den Waschvorgang startet.

Die Waschmaschine soll abhängig von den Eingabewerten eine bestimmte Dauer in Betrieb sein und eine bestimmten Energiemenge verbrauchen. Ist die Waschmaschine einmal gestartet, so soll es nicht möglich sein, den Waschvorgang vorzeitig zu beenden. Die Waschmaschine soll stattdessen gezwungen sein, bis zum Ende durchzulaufen. Dies soll ein versehentliches Unterschlagen von Waschzyklen vermeiden.

Der Energieverbrauch soll sich in jedem Zeitintervall verändern. Dabei steht es nicht im Vordergrund, eine möglichst realistische Implementierung einer Waschmaschine zu entwerfen, sondern eine sinnvolle Komponente für die Simulation zu schaffen. Deshalb wird die Waschmaschine auch keinen Eingabewert haben, der für die Simulation verschiedener Betriebsmodi zuständig ist.

Da die Waschmaschine keinem bestimmten Modell einer reell existierenden Waschmaschine folgt, dienen als Richtwerte für den Stromverbrauch eines Waschvorgangs die von der Webseite [55] entnommenen Werte. Die Waschmaschine hat eine maximale Beladungskapazität von 10 kg und soll Waschtemperaturen von 30 Grad Celsius bis 60 Grad Celsius anbieten. Dabei entspricht der Stromverbrauch unter Volllast bei 60 Grad Celsius etwa 1,0 kWh und die Dauer eines Waschzyklus ist 120 Minuten, was acht Zeitintervallen entspricht. Ist die Maschine nur teilbeladen (5kg), so sinkt der Stromverbrauch auf 0,85 kWh und die Waschdauer auf 105 Minuten. Bei abnehmender Beladung und Waschtemperatur sollen auch Dauer eines Waschvorgangs und die benötigte Energiemenge abnehmen. Mindestens aber dauert ein Waschvorgang 40 Minuten und verbraucht etwa 0,5 kWh. Die Grafiken 5.11 und 5.12 veranschaulichen den Energiebedarf sowie die Betriebsdauer der Waschmaschine in Abhängigkeit von der Beladung und der Waschtemperatur.

Damit der Stromverbrauch nicht in jedem Zeitintervall gleich ist, wird dem Verbrauch eines Waschzykluses eine kurvenförmige Darstellung über die Zeit unterstellt. Da es für die Simulation ausreichend ist, steigt dabei der Verbrauch für die Hälfte aller Zeitschritte linear an und fällt anschließend wieder ab.

5.6.5 Zweiter steuerbarer Verbraucher

Der zweite steuerbare Verbraucher sollte ein Gerät sein, welches mehr Leistung erbringen kann, als der erste steuerbare Verbraucher, die Waschmaschine. Eine Komponente, die mehr Leistung abrufen kann, fügt sich besser in unser Szenario ein. Geplant war von Beginn an, sich eher auf Gebäude in der Größenordnung eines

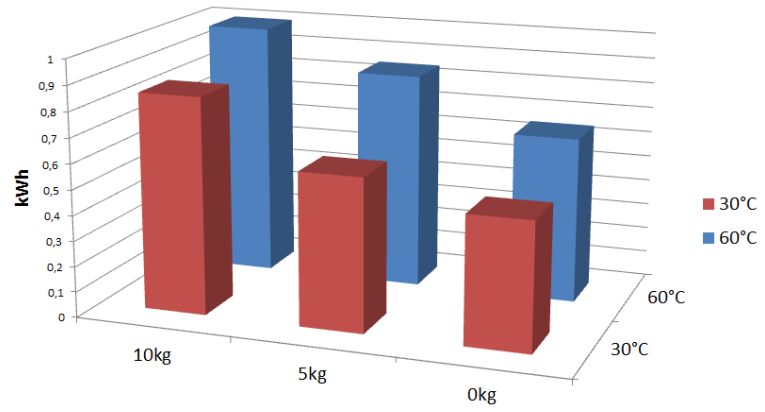


Abb. 5.11 Energieverbrauch der Waschmaschine in kWh

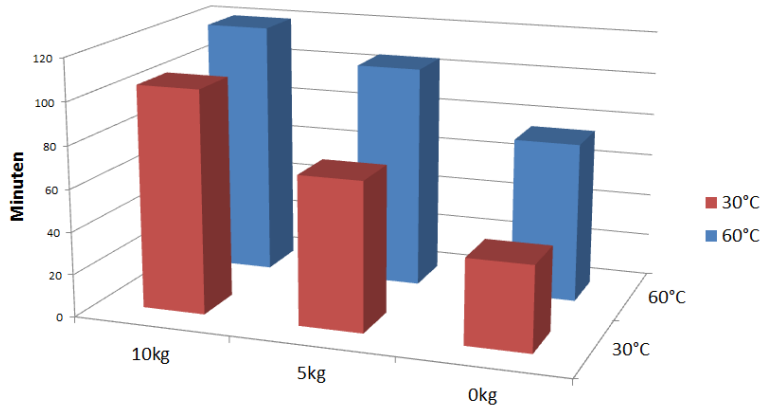


Abb. 5.12 Betriebsdauer der Waschmaschine in Minuten

Hotels oder Krankenhauses anstatt eines Einfamilienhauses zu konzentrieren. Von daher hat eine einzelne Waschmaschine nur ein geringes Potenzial, gewinnbringend in der Optimierung gesteuert zu werden. Ein für unser Szenario besser geeigneter Verbraucher schien uns deshalb eine Klimaanlage oder ein Kühlhaus zu sein. Beide Anlagen sind gemeinhin in Hotels oder Krankenhäusern zu finden. Nach einer kurzen Recherche wurde entschieden, das Kühlhaus als Modell umzusetzen. Die Simulation des Verhaltens des Kühlhauses mit einem Zweipunktregler erschien dem Autor intuitiver als die Klimaanlage, bei der das Verhalten der Nutzer, die die Raumtemperatur in ihrem Zimmer individuell einstellen können, noch in die Simulation mit einzubauen gewesen wäre.

5.6.5.1 Grundlegende Arbeitsweise

Wie angesprochen arbeitet das Kühlhaus mit einem Zweipunktregler. Dazu wird dem Modell eine Zieltemperatur, auf die die Kühlkammer herunter gekühlt werden soll, sowie eine Hysterese vorgegeben. Die Hysterese ist die Toleranz zur Zieltemperatur, während der nicht erneut gekühlt werden muss. Beide Angaben werden in ∘C gemacht. Intern rechnet das Modell jedoch mit Kelvin. Dies ist der Kompatibilität zu den physikalischen Formeln geschuldet, die für die Berechnung der Wärmeverluste und Kühlleistung eingesetzt werden und später noch genauer erläutert werden.

Wie stark die Temperatur in der Kühlzelle gesenkt werden kann, liegt besonders an dem verbauten Kühlaggregat. Je stärker das Aggregat, desto mehr Wärmeenergie kann dem Kühlraum entzogen werden. In dem Beispielszenario wird das Modell mit einem 1,8 kW starkem Kühlaggregat betrieben. Das Modell arbeitet als Luftkühler. Das bedeutet, dass die Luft der Kühlzelle in das Aggregat gelangt, dort abgekühlt wird und anschließend in die Kühlzelle zurückgeführt wird. Dabei muss beachtet werden, dass die Kühlleistung des Aggregates von seinem Wirkungsgrad abhängt. Da es sich um eine elektrische Pumpe handelt, gibt es bereits einen Wirkungsgrad der Anlage selber, der das Verhältnis von Energie, die zur Kühlung verwendet wurde, und insgesamt aufgenommener Energie angibt. In unserem Beispielszenario geben wir diesen Wirkungsgrad mit 60% an. Zusätzlich sinkt der Wirkungsgrad, wenn die Differenz zwischen der Temperatur der angesaugten Luft und der minimalen Kühltemperatur kleiner wird.

Ein zweiter Punkt, der die Temperaturabsenkung mit bestimmt, ist die Befüllung des Kühlraumes. Die kalte, aus dem Kühlaggregat ausströmende Luft kühlt die in der Kühlzelle gelagerten Stoffe ab. Je voller das Kühlhaus befüllt ist, desto langsamer wird das Kühlhaus abgekühlt. Vereinfachend wird dabei angenommen, dass das Kühlhaus mit einem Stoff gefüllt wird. In diesem Modell handelt es sich dabei um Wasser. Das Beispielszenario arbeitet mit einem Füllgrad von 70%.

Neben des Kühlverhaltens werden in diesem Modell auch die Wärmeverluste modelliert. Neben Höhe, Breite und Länge des Innenraumes der Kühlzelle benötigt das Modell weitere Angaben zur Dicke und Wärmeleitfähigkeit der Wärmedämmung. Damit wird der Wärmeverlust in Abhängigkeit zur Außentemperatur berechnet. Es wird keine Unterscheidung der einzelnen Wände vorgenommen, ob dahinter zum Beispiel ein Innen- oder Außenraum oder der Boden ist. Der Wärmeverlust führt zu einer Erwärmung des Innenraumes. Steigt die Temperatur über den eingestellten Zielwert und die mögliche Abweichung, springt das Kühlaggregat an und kühlt das Kühlhaus.

5.6.5.2 Mathematische Modellierung

Das Modell beginnt in einem Durchlauf damit, die Wärmeverluste zu berechnen, um daraus den Anstieg der Temperatur zu gewinnen. Ist die neue Raumtemperatur berechnet, kann der Energiebedarf berechnet werden, der aufgewendet werden muss, um das Kühlhaus wieder abzukühlen. Daraus ergibt sich anschießend die Zeit, die

das Kühlaggregat zur Kühlung benötigt, und somit auch die für die anderen Systeme benötigten „verbrauchten“ Kilowattstunden elektrische Energie.

Berechnung der Innentemperatur

Um die Wärmeverluste zu berechnen, wird die Fläche benötigt, die die Kühlzelle gegen die Umgebungstemperatur abschirmt. Dazu wird aus der Höhe h, der Breite b und der Länge l die Oberfläche der Innenwände berechnet:

$$A = (l \cdot b \cdot 2) + (b \cdot h \cdot 2) + (l \cdot h \cdot 2); \tag{5.9}$$

Beachtet werden muss bei den Daten h, l und b, dass es sich hierbei um das Innenmaß des Kühlraumes handelt. Darauf aufbauend kann der Wärmeverlust berechnet werden [8]. Dieser ist bei einer ebenen Wand gleichzusetzen mit dem Wärmestrom $\dot{Q}$ durch das Dämmmaterial bei gegebener Temperaturdifferenz.

$$\dot{Q} = \frac{\lambda}{s} \cdot A \cdot \Delta T \tag{5.10}$$

Dieser Wärmestrom $\dot{Q}$, der in Watt angegeben wird, wird nun in Joule umgerechnet. Ein Joule ist dabei eine Wattsekunde.

$$\dot{Q}_J = \dot{Q} \cdot 15 \text{ Minuten} \cdot 60 \text{ Sekunden} \tag{5.11}$$

Die Umrechnung in Joule ist nötig, da die neue Raumtemperatur in Ws/K berechnet wird. Die Multiplikation mit 15 Minuten ist notwendig, da ein Zeitschritt in der Simulation 15 Minuten beträgt. Anschließend wird noch berechnet, um wie viel Grad die in dem Kühlhaus gelagerten Dinge erwärmt werden. Das Modell verwendet hier Wasser und Luft als in dem Kühlhaus befindliche Stoffe. Mit dem Parameter „filling level“ lässt sich der Füllstand mit Wasser einstellen. Zunächst wird die Wärmekapazität des Raumes C_R berechnet.

$$C_R = ((f \cdot V \cdot D_W) \cdot C_W) + (((1 - f) \cdot V \cdot D_A) \cdot C_A); \tag{5.12}$$

In der Formel 5.12 steht f für den Füllgrad des Lagerhauses in Prozent mit Wasser und $1 - f$ gibt den Anteil der Luft an. V ist das Volumen des Kühlhauses und D_W steht für die Dichte des Wassers, sodass die vorhandene Masse des Wasser berechnet wird. Dass sich die Dichte des Wasser mit unterschiedlichen Temperaturen ändert wird in dieser Formel nicht beachtet. Es wird eine Dichte von $1000 \frac{kg}{m^3}$ angenommen. Anschließend wird es mit der spezifischen Wärmekapazität des Wassers multipliziert. Die selbe Rechnung wird auch für die vorhandene Luft durchgeführt, wobei eine Dichte von $1,2041 \frac{kg}{m^3}$ angenommen wird. Die Addition der Ergebnisse ergibt die Energie, die benötigt wird, um den Raum um einen Kelvin zu erwärmen. Die Einheit ist $\frac{Ws}{K}$

Anschließend wird aus dem Wärmestrom $\dot{Q}$ der in die Kühlzelle strömt und der Wärmekapazität des Raumes C_R der Temperaturverlust in K berechnet:

$$T_{loss} = \frac{\dot{Q}}{C_R} \tag{5.13}$$

Anschließend wird aus der Innentemperatur und der Erwärmung des Raumes die neue Innentemperatur berechnet. Liegt diese unter der eingestellten Zieltemperatur inklusive Hysterese, ist das Kühlhaus noch kalt genug und es muss nicht gekühlt werden. Wird die Zieltemperatur jedoch zu stark überschritten, springt das Kühlaggregat an.

Berechnung der Kühlleistung

Die Simulation der Kühlung der Kühlzelle arbeitet zunächst auf der Basis der Energie, die dem Raum entzogen werden muss. Daraus wird anschließend die Leistungsaufnahme des Kühlaggregates hergeleitet. Zunächst wird die Energie berechnet, die benötigt wird, um den Innenraum auf die gewünschte Temperatur inklusive Hysterese H herunter zu kühlen. Die benötigte Energie E_D wird berechnet, indem die Wärmekapazität des Raumes pro Kelvin C_R mit der Temperaturdifferenz aus aktueller Temperatur T_a und gewünschter Temperatur T_t multipliziert wird.

$$E_D = C_R * (T_a - (T_t - H)) \tag{5.14}$$

In einer zweiten Rechnung wird die Energie berechnet, die von dem Kühlaggregat aufgebracht werden wird, wenn sie 15 Minuten auf maximaler Leistung arbeitet. Die Energie, die zur Kühlung genutzt werden kann hängt von der Leistung des Kühlaggregates und der Effizienz mit der es arbeitet ab. Wie effektiv das Kühlaggregat ist, wird durch den Wirkungsgrad η und die Temperaturdifferenz zwischen angesaugter und abgegebener Luft bestimmt [52]. Wir haben zur Bestimmung der Effektivität einen linearen Abfall des Wirkungsgrades angenommen, der sich an folgender Quelle anlehnt [17, Seite 8] und folgendermaßen berechnet wird:

$$\eta_{all} = \frac{\Delta T}{\Delta T_{opt}} \cdot \eta_{el} \tag{5.15}$$

In diesem Fall nehmen wir an, dass ΔT_{opt} 25 ∘ C beträgt. Ist die Temperaturdifferenz größer, wird der Wirkungsgrad der Anlage nicht verringert. Aufgrund dieser Effizienzrechnung wird anschließend die Energie $E_{cooling}$ der Viertelstunde in Ws aus der Leistung P in kWh berechnet, die zur Kühlung in den Raum eingebracht wird.

$$E_{cooling} = P \cdot 1000 \cdot 15 \cdot 60 \cdot \eta_{all} \tag{5.16}$$

Ist der Energiebedarf E_D größer als die Energie die zur Kühlung bereitgestellt werden kann $E_{cooling}$, wird das Kühlaggregat die gesamte Viertelstunde arbeiten und die maximale Leistung erbringen. Ist E_D kleiner als $E_{cooling}$, wird ausschließlich die

benötigte Energie dieser Viertelstunde in *kWh* ausgegeben. Anschließend ist die Raumtemperatur die Zieltemperatur inklusive Hysterese.

5.7 Samplingkonzept

Eine wichtige Entwurfsentscheidung ist, inwiefern die Optimierung Informationen der EVS-Komponenten in die Optimierung einbezieht. Um überhaupt sinnvoll optimieren zu können, muss die Optimierung beispielsweise wissen, wie viel Leistung die Komponenten erzeugen oder verbrauchen, ob der Wärmebedarf des Gebäudes gedeckt ist und ob die Fahrpläne valide sind, d.h. keine Gerätebedingungen der EVS-Komponenten verletzen.

Das Problem ist, dass diese Informationen ohne weiteres der Optimierung nicht zur Verfügung stehen, da viele Informationen nur in den Modellen bekannt sind. Eine mögliche Lösung ist, dass im Energiemanagementsystem Kopien der EVS-Komponenten zusätzlich simuliert werden und diese Kopien der Optimierung die nötigen Informationen bereitstellen. Diese Lösung ist aber nur mit großem Aufwand umsetzbar, da für jede EVS-Komponente eine zusätzliche Kopie im EMS programmiert werden müsste. Neben den enormen Aufwand ist diese Lösung auch nicht generisch, da für eine Erweiterung des Systems um weitere EVS-Komponenten auch ein Modell der Komponenten für die Optimierung geschrieben werden müsste.

Eine gute Möglichkeit, um die Umsetzung generischer zu machen, ist ein Samplingkonzept umzusetzen. Das bedeutet, dass nicht direkt Informationen, wie z.B. interne Gerätebedingungen, an die Optimierung gegeben werden, sondern die EVS-Komponenten selbst mögliche Fahrpläne erstellen und nur Samples mit erzeugter und verbrauchter elektrischer und thermischer Leistung an die Optimierung übergibt, welche dann versucht, eine möglichst gute Kombination von Samples als Fahrplan zu erstellen. Dadurch wird eine Divide-And-Conquer Strategie umgesetzt, bei der die Aufgaben der Optimierung in die Erstellung der Samples und Auswahl der Samples örtlich und logisch getrennt werden. Diese Trennung vereinfacht auch die Struktur der Kommunikation, denn zwischen diesen beiden Teilen müssen nicht beliebige Informationen über Gerätebedingungen, verschiedenen Parametern, Leistungen etc. verteilt werden, sondern die Optimierung im Energiemanagementsystem muss nur Samples von den EVS-Komponenten abrufen können. Dabei kann die Struktur der Samples für die Komponenten gleich bleiben, denn Samples sind nur Leistungs- oder Energieinformationen zu verschiedenen Zeitpunkten und somit als einfache Arrays oder Vektoren umsetzbar. Dies verbessert die Erweiterbarkeit enorm, da bei höherer Anzahl der EVS-Komponenten die Optimierung nur mehr Samples abrufen muss, diese aber keine neuartigen Informationen darstellen.

Neben diesen Vorteilen existiert jedoch das Problem, dass die EVS-Komponenten bzw. unsere Modelle der EVS-Komponenten Samples erzeugen können müssen. Dies bedeutet einen erhöhten Programmieraufwand für die EVS-Komponenten. Dieser Aufwand ist aber akzeptabel, da die oben genannten Vorteile an anderen Teilen des Systems wie beschrieben enormen Aufwand einsparen.

5.7.1 Customer Energy Manager

Eine grundlegende Frage beim Entwurf des Samplings ist, an welcher Stelle diese generiert werden sollen. Eine Möglichkeit ist, diese Samples direkt in den EVS-Komponenten zu erzeugen, wodurch die Kommunikation zwischen der Komponente und dem Code zum Generieren der Samples quasi nicht vorhanden ist. Jedoch hätte diese Lösung den enormen Nachteil, dass die Komplexität der Matlabmodelle für unsere EVS-Komponenten steigt und schlimmer noch dies für reale EVS-Komponenten gar nicht möglich ist, da dafür der interne Programmcode der EVS-Komponenten änderbar sein müsste.

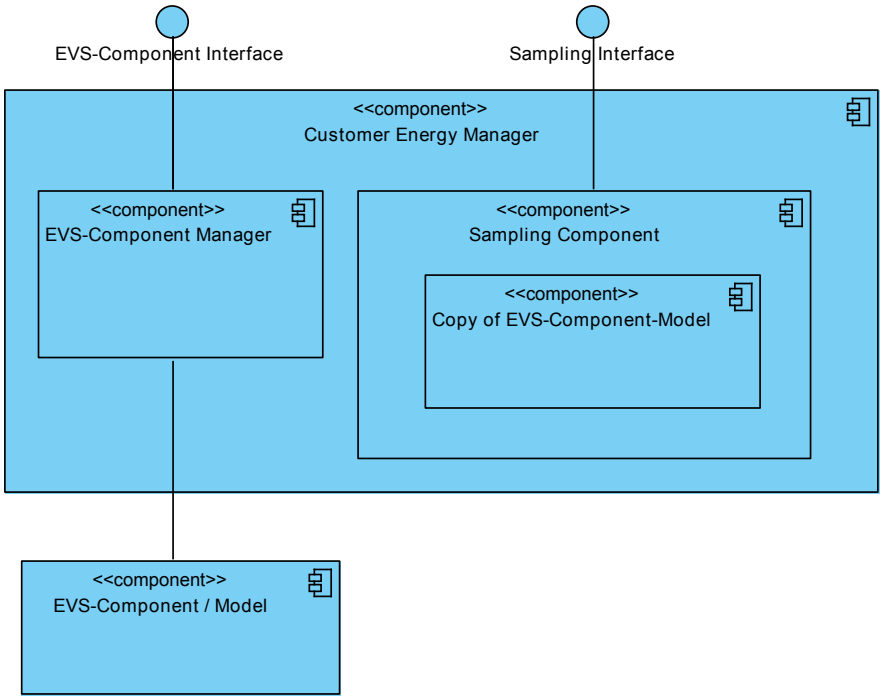


Abb. 5.13 Architektur eines Customer Energy Managers mit Sampling

Eine sinnvolle Umsetzung ist nur dann möglich, wenn eine Trennung zwischen Sampling und der eigentlichen EVS-Komponente gewährleistet bleibt. Beispielsweise indem man das Sampling in einer Zwischenschicht zwischen EVS-Komponente und unseren System realisiert. Dazu haben wir uns an einem Entwurf des Europäischen Komitees für elektrotechnische Normung (CENELEC) orientiert. [48] Dieser Entwurf sieht für den Zugriff aus EVS-Komponenten einen Customer Energy Manager (CEM) vor, welcher als zusätzliche Schicht die Verwaltung und Steuerung einer

EVS-Komponente übernimmt, so dass ein direkter Zugriff nicht mehr möglich ist. So ein CEM bietet äußeren Systemen Schnittstellen an, über welche die Verwaltung der Geräte möglich ist. Eine mögliche Schnittstelle kann Zugriff auf einen sogenannten Flexibility Operator bieten, über welchen äußere Systeme Flexibilitäten von EVS-Komponenten, d.h. Möglichkeiten von Leistungsänderungen, abrufen können.

Ein solcher Flexibility Operator ist beispielsweise auch eine Samplingkomponente, da sie dem Energiemanagementsystemen einen möglichen Rahmen erlaubter Leistungsänderungen vorgibt. Der Entwurf unseres Customer Energy Manager ist in Abbildung 5.13 zu sehen. Der Custom Energy Manager trennt sich dabei in einen Manager der EVS-Komponente, der zur Verwaltung der EVS-Komponente bzw. Ausführung der simulierten EVS-Komponente dient und einer Samplingkomponente, welche Samples generiert und für das EMS zur Verfügung stellt. Dabei ist anzumerken, dass die Samplingkomponente zur Generierung der Samples Wissen über die EVS-Komponente benötigt. Im Fall der simulierten EVS-Komponenten kann dieses Wissen im Form einer identischen Kopie der simulierten EVS-Komponente vorliegen. Mit Hilfe dieser Kopie kann die Samplingkomponente Simulationsdurchläufe durchführen und testen, wie sich die EVS-Komponente bei unterschiedlichen Samples verhält.

Ein Algorithmus, mit der die Samplekomponente Samples erzeugen kann, ist in [3] beschrieben. Dieser Algorithmus wählt Steuerungsparameter zufällig und überprüft anschließend, welches Ergebnis das Modell liefert. Ist das Ergebnis valide, wird das Ergebnis für diesen Zeitschritt gespeichert und es werden neue Parameter für den nächsten Schritt erzeugt. Ist das Ergebnis jedoch nicht valide, muss der letzte Schritt des Modells rückgängig gemacht werden und mit anderen Parametern wiederholt werden. Dieses Vorgehen wird so lange wiederholt bis ausreichend valide Schritte für ein Sample erfolgt sind.

5.7.2 Steuerung des Customer Energy Managers

Im Folgenden wird beschrieben, wie der Customer Energy Manager von außen gesteuert wird. Die Steuerung erfolgt dabei über Flags, die über die Schnittstelle zum Custom Energy Manager zugreifbar sind. Dabei gibt es vier Flags:

- **SF-Init-Flag:** Dieses Flag wird vom SF auf TRUE gesetzt, sobald es alle Parameter dem CEM mitgeteilt hat, welche die EVS-Komponente benötigt. Der CEM überträgt diese Parameter dann an die EVS-Komponente und der Sampling-Komponente und setzt nach der Initialisierung das Flag wieder auf FALSE.
- **EMS-Flag:** Dieses Flag wird vom EMS auf TRUE gesetzt, sobald es dem CEM mitgeteilt hat, welches Sample, d.h. in diesem Fall welche Steuerungssignale, die EVS-Komponente für den nächsten Zyklus benutzen soll. Nach einem simulierten Zyklus setzt der CEM das Flag wieder auf FALSE.
- **SF-Flag:** Dieses Flag wird vom SF auf TRUE gesetzt, sobald es alle Werte dem CEM mitgeteilt hat, welche die EVS-Komponente zur Simulation eines Zyklus

benötigt. Zu solchen Werten zählen beispielsweise Informationen über das Wetter. Nach einem simulierten Zyklus setzt der CEM das Flag wieder auf FALSE.

- **Sampling-Flag:** Dieses Flag wird auf TRUE gesetzt, sobald die Sampling-Komponente Samples erzeugen soll. Nach Fertigstellung der Samples wird das Flag wieder auf FALSE gesetzt.

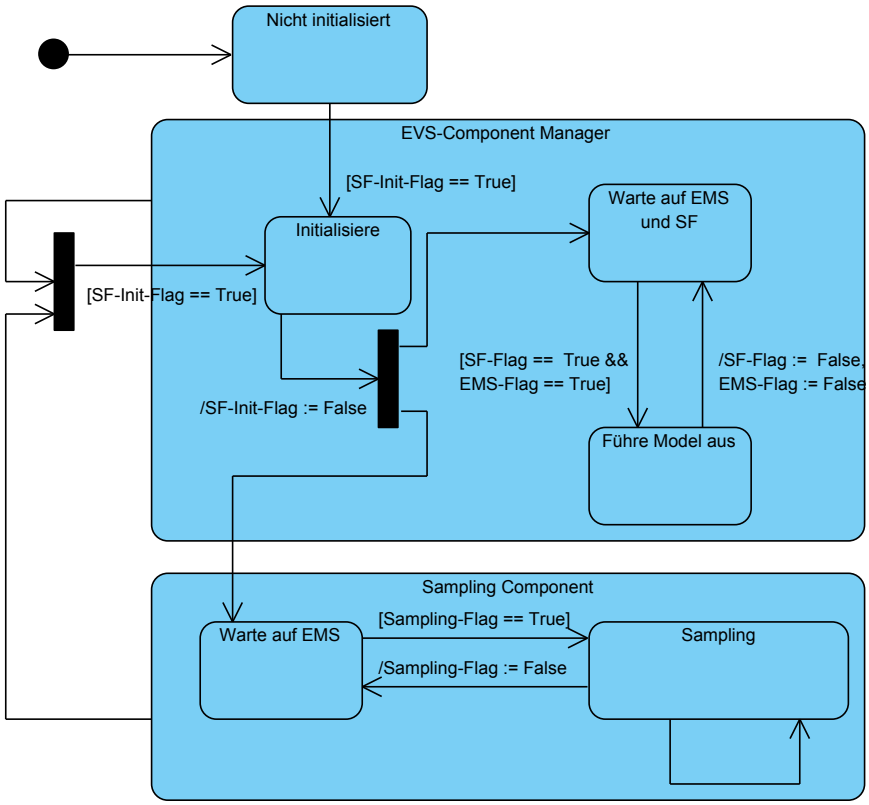


Abb. 5.14 Zustandsdiagramm des Customer Energy Managers

Dieser Ablauf ist in Abbildung 5.14 bildlich dargestellt und beschreibt die Zustände, die der CEM durchläuft. Die Realisierung eines Custom Energy Managers findet sich in Abschnitt 6.7.

5.8 Model-Manager

Der Model-Manager dient innerhalb des Simulationsframeworks der Verwaltung und dem einfachen Zugriff auf die Daten des Datenmodells, welches der Repräsentation des physischen Umfelds der Simulation dient (siehe Abschnitt 5.3). Aus der Abstraktion des Datenmodells ergeben sich Datensätze, die durch eine unbekannte Anzahl von Attributen, die wiederum Werte repräsentieren, gekennzeichnet sind. Diese vorgegebene Hierarchieordnung ermöglicht für Datensätze das Mapping einer Semantik zu dem entsprechenden Attribut.

Aus dem Grund, dass aus unterschiedlichen Datenquellen (Wetterdaten, Grundlast, etc.) jeweils eigene Datensätze generiert werden, besteht eine unbekannte Anzahl von Datensätzen. Durch die dynamische Anzahl der Datensätze wird eine Zuordnung der in dem Datensatz vorhandenen Semantiken zu dem Datensatz selbst benötigt. Dementsprechend wird ein übergeordneter Datencontainer benötigt, der die Zuordnungen der Semantiken zu den Datensätzen gleichermaßen handhabt wie die Zuordnung der Datensätze zu den Attributen. Daraus resultiert eine Baumstruktur, die idealerweise durch das *Composite Pattern* realisiert werden kann.

Composite Pattern

Das *Composite Pattern* ist ein Entwurfsmuster um Teil-Ganzes-Hierarchien zu repräsentieren, indem Objekte zu Baumstrukturen zusammengefügt werden. Das *Pattern* ermöglicht die einheitliche Verwendung von Objekten und ihren Behältern [15, S. 163 ff.].

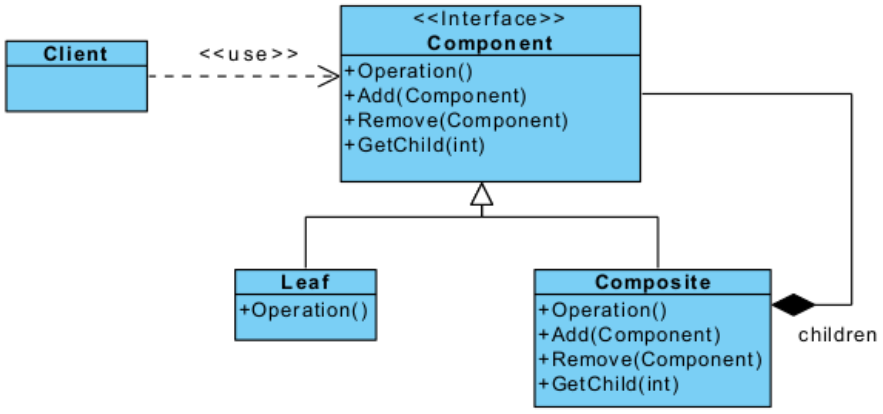


Abb. 5.15 Darstellung des Composite Patterns

Abbildung 5.15 zeigt den beispielhaften Aufbau dieses Entwurfmusters, das über die abstrakte Klasse *Component* die einheitliche Verwendung der Klassen *Leaf* und *Composite* erlaubt. *Leaf* stellt dabei eine Implementation der *Component*-Klasse dar. Das Besondere dieses Entwurfsmusters liegt dabei im Entwurf der *Composite*-Klasse. Während diese selbst die *Component*-Klasse implementiert und somit *Component* als übergeordneten Typ besitzt, können ihr weitere Objekte vom Typ Component zugeordnet werden. Die Implementation der durch Component definierten Funktionen müssen dabei die Funktionen der Component-Instanzen berücksichtigen, die dem Composite zugeordnet wurden.

Verwendung des Composite Patterns für das Datenmodell

Die einzelnen Attribute des Datenmodells mit den zugehörigen Werten stellen die *Leafs* des *Patterns* dar. Diese sind im Folgenden als *DataModel* definiert. Sowohl die Datensätze, als auch die Datencontainer sind Behälter, die eine beliebige Anzahl von *Components* zusammenfassen. Dementsprechend werden diese auf das *Composite* des *Patterns* abgebildet. In der entworfenen Struktur werden die *Composites* als *ModelManager* bezeichnet.

Für den Zugriff auf die Daten ist ein konkreter Ansprechpartner zur Kommunikation von *SIMController* und *SIMModel* notwendig, der den Zugriff auf die einzelnen Objekte des Baums ermöglicht. Daher wird als Wurzel des Baums ein *ModelManager* erstellt, der mit der Struktur befüllt wird, die aus dem Datenmodell resultiert.

5.9 Systemkommunikation

Für die Simulation ist es wichtig, dass das Energiemanagementsystem und das Simulationsframework jeweils bidirektional mit der Steuerungsebene kommunizieren können. Einerseits sollen Steuersignale von der SE übertragen werden, z.B. um einen Optimierungslauf zu starten (vgl. Abschnitt 5.9.1). Andererseits ist auch eine Rückmeldung des jeweiligen Systems sinnvoll, um z.B. Fehler in der SE anzeigen zu können. Dazu ist es zunächst wichtig, ein Grundkonzept zu entwickeln, auf welchem Weg die Kommunikation ablaufen soll.

5.9.1 Entwurf der Kommunikation von Steuersignalen

Sowohl in den jeweiligen Systemen (SF bzw. EMS) als auch in der Steuerungsebene gibt es eine Reihe von Anforderungen, die den Austausch von Steuersignalen und Daten zwischen der Steuerungsebene bzw. einer dort implementierten GUI sowie dem jeweiligen System notwendig machen. So besagt z.B. die Anforderung SF-60, dass das Szenario initialisiert werden muss oder die Anforderung SE-EMS-05,

dass das EMS bzw. dessen Optimierung über die Steuerungsebene gestartet werden muss. Ferner soll das EMS gemäß EMS-170 auch wieder aus der SE beendet werden können. Ebenfalls ist es wichtig, dass die Kommunikation auch funktioniert, wenn ein System nicht antwortet, z.B. wenn das EMS ohne das Simulationsframework gestartet wird.

Zunächst sollte evaluiert werden, auf welchem Weg die Kommunikation erfolgen soll. Einerseits wäre es möglich, dass sich die Steuerungsebene einzeln mit beiden Systemen verbindet. In diesem Fall würden die Steuerungsebene über getrennte Wege Anfragen stellen, die von den Systemen beantwortet werden. Diese Lösung wäre relativ einfach gehalten, hätte aber den Nachteil, dass bei Hinzukommen eines weiteren Systems oder einer weiteren Kommunikationsstruktur ein hoher Implementierungsaufwand nötig wäre. Zudem müsste die Steuerungsebene die Adresse jedes Kommunikationspartners kennen. Vorteilhaft wäre hingegen, dass bei Ausfall der Kommunikation zu einem System die anderen Systeme trotzdem noch angesprochen werden könnten.

Eine weitere Möglichkeit stellt ein *Webservice* dar. Bei dessen Verwendung würden sich die Systeme bei einem Service-Broker anmelden, der wie eine Art Telefonbuch die Adressen der Systeme verwaltet (i.d.R. als Uniform Ressource Identifier (URI)). Gleichzeitig sollten die Systeme dem Service-Broker Informationen über ihre Schnittstelle, z.B. im XML-Format mitteilen. Die Steuerungsebene könnte dann beim Service-Provider die Adressen und die Informationen über die Schnittstellen der jeweiligen Systeme abrufen. Anschließend könnten die Systeme kommunizieren. Es würde sich somit eine sehr flexible und erweiterbare Architektur ergeben. Bei Ausfall des Webservices könnten bestehende Verbindungen dennoch weiter kommunizieren. Nachteilig an dieser Lösung wäre der zusätzliche Aufwand für das Definieren der Schnittstellen und bei Verwendung von XML der zusätzliche Rechenaufwand das Parsen der XML-Beschreibungen. Zudem wären zusätzliche Fremdbibliotheken notwendig. (vgl. [29], [61])

Soll der Aufwand der Schnittstellendefinition, zusätzliche Fremdbibliotheken sowie der Rechenaufwand durch das Parsen eingespart werden, ergibt sich eine dritte Möglichkeit (vgl. Abbildung 5.16): Das zentrale Element des Entwurfs stellt die Broker-Klasse dar. Diese Klasse soll ähnlich wie ein Postamt funktionieren: Werden Steuersignale an diese Klasse versendet und mit einem Empfänger (z.B. EMS) versehen, besteht die Aufgabe darin, diese Nachricht dem Empfänger korrekt zuzustellen und ggf. gesendete Antworten ebenfalls zurückzuleiten. Es soll somit eine bidirektionale Kommunikation ermöglicht werden. Dies erfordert, dass sich die Systeme zuvor beim Broker registrieren, damit deren Identität und Anwesenheit bekannt ist. Die Systeme können sich somit ähnlich wie beim Webservice an den Broker wenden - anstatt jedoch eine Adress- bzw. Serviceanfrage zu senden, wird die Nachricht um den Empfänger ergänzt, während der Broker für die Zustellung verantwortlich ist. Durch die geplante Registrierung der Systeme wird eine Erweiterbarkeit des Brokers sichergestellt: So könnten etwa für das EMS und das SF eigenständige Zeitgeber realisiert werden, die sich über den Broker synchronisieren (vgl. auch Abschnitt 5.12). Jedes System hat über den Broker die Möglichkeit, mit jedem anderen System zu kommunizieren. Nachteilig an dieser Lösung ist, dass z.B. die Steuerungsebene wis-

sen muss, welche Nachrichten sie an die Systeme senden muss, da keine Definition der Schnittstellen abgerufen werden kann. Zudem könnte bei Ausfall des Brokers keine Kommunikation mehr stattfinden.

Insgesamt wird die dritte Variante bevorzugt, da sie eine ausreichende Erweiterbarkeit sicherstellt und den Aufwand gegenüber einem Webservice verringert. Die Broker-Klasse sollte im EMS beheimatet sein, da der Zeitgeber des EMS immer benötigt wird und das EMS somit immer verfügbar ist (vgl. Abschnitt 5.12). Die Systemkommunikation soll jedoch austauschbar gestaltet werden, um z.B. eine spätere Ansiedlung im SF zu ermöglichen.

Bei Umsetzung des in Abbildung 5.16 dargestellten Entwurfs wird neben dem Broker in den Systemen (im Beispiel EMS) eine Klasse für den Empfang von Steuersignalen benötigt. Dazu kann ein gemeinsames Interface (*ControlSignalSystemCommunication*) genutzt werden, das im EMS und im SF implementiert wird (vgl. auch Abbildung 5.17). Dort sollen jeweils die an das System gerichteten Steuersignale empfangen und beantwortet werden. Da ständig Nachrichten eintreffen können und auf diese gewartet werden muss, sollten die Implementierungen dieses Interfaces in einem eigenen Thread ausgeführt werden. Daher wird eine zweite Klasse benötigt (*iCommunication*), die von der *ControlSignalSystemCommunication* über eingehende Signale informiert wird und z.B. den Controler informieren kann.

Für das Versenden der Steuerbefehle aus der Steuerungsebene wird ebenfalls eine Klasse benötigt. Um ggf. zukünftig notwendige, differenzierte oder zu unterschiedlichen Zeitpunkten versendete Nachrichten an beide Systeme versenden zu können, bieten sich zwei Klassen an (*EMSCommunication* und *SIMCommunication*). Dies ermöglicht, dass etwa das Simulationsframework ohne das EMS betrieben wird (vgl. SF-80A). Dazu muss lediglich der Broker des EMS erreichbar sein. Die beiden Klassen sollten wiederum ein gemeinsames Interface nutzen (*iCLCommunication*, vgl. Abbildung 5.16 und Abbildung 5.17). Benötigt werden vier verschiedene Signale:

1. Ein Initsignal: Die Steuerungsebene sollte die Systeme über ein neues Szenario bzw. einen neuen Run informieren. Dazu kann ein Initsignal verwendet werden.
2. Ein Startsignal: Die SE informiert die Systeme, dass der Taktgeber einen neuen Lauf starten soll.
3. Ein Stopsignal: Die SE informiert die Systeme über die Beendigung des aktuellen Laufs.
4. Ein Statusabfrage: Die SE prüft, ob die Systeme erreichbar sind.

Die Verwendung von Interfaces ermöglicht eine komplette Austauschbarkeit der Implementierung der Systemkommunikation inklusive der Broker-Klasse.

Die bidirektionale Kommunikation ist notwendig, damit Steuersignale (z.B. ein Startsignal) von der Steuerungsebene an die Systeme gesendet werden können und im Anschluss eine Rückmeldung erfolgt, z.B. dass der Start des Systems erfolgreich durchgeführt wurde.

Zusätzlich sollte eine unidirektionale Kommunikation möglich sein: Die Systeme müssen nach Ablauf eines Taktes die Steuerungsebene informieren (vgl. CYCLE-Signal in Abbildung 5.16), damit etwa Komponentendaten in der GUI aktualisiert werden können. Dies kann mit einem Observerpattern kombiniert werden (vgl. Ab-

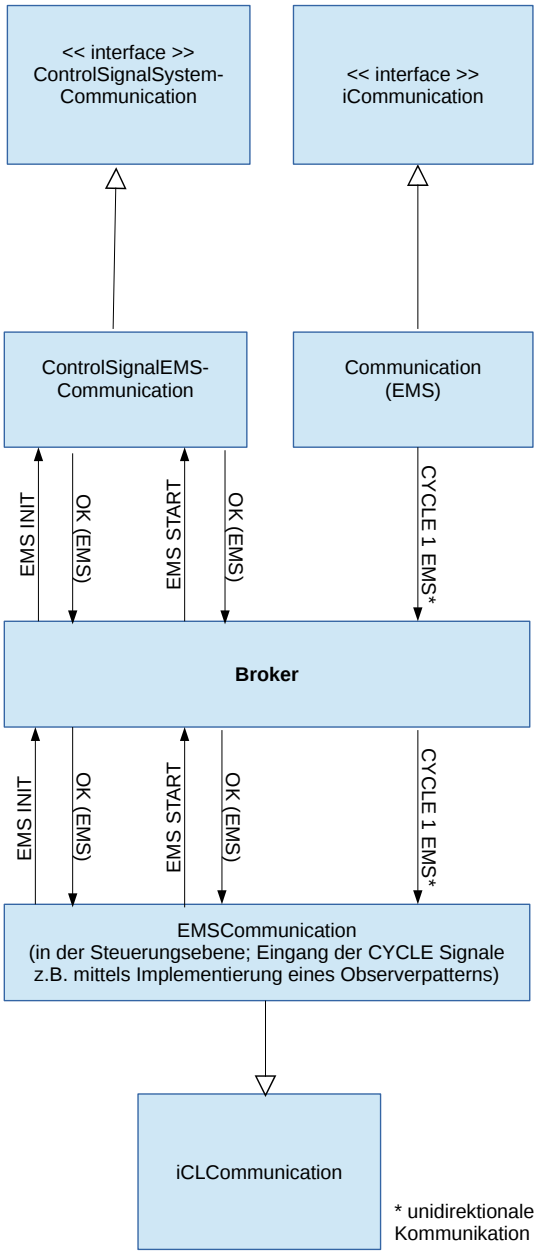


Abb. 5.16 Entwurf der Architektur der Kommunikation von Steuersignalen (Eigene Darstellung)

bildung 5.18): Nach Ablauf eines Taktes informiert z.B. die EMS-spezifische Implementierung von *iCommunication* die Steuerungsebene darüber, ohne dass auf eine Antwort gewartet werden muss (daher wird kein zusätzlicher Thread benötigt). Die

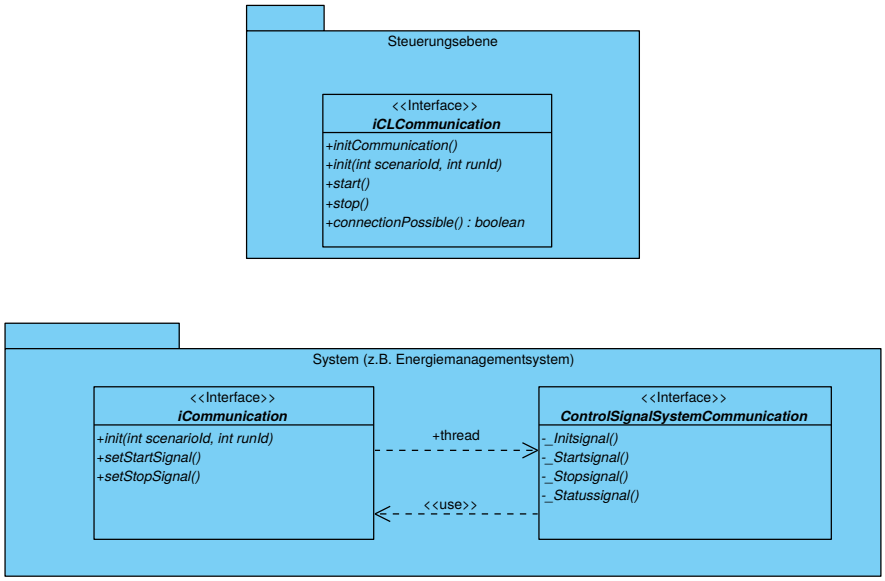


Abb. 5.17 Klassendiagramm der Architektur zur Übermittlung von Steuersignalen (Eigene Darstellung)

Steuerungsebene wiederum implementiert ein Observable, bei der sich der Controller, der für die GUI zuständig ist, registriert. Der Controller implementiert somit den zugehörigen Observer und kann bei Eingang eines Taktsignals die GUI informieren.

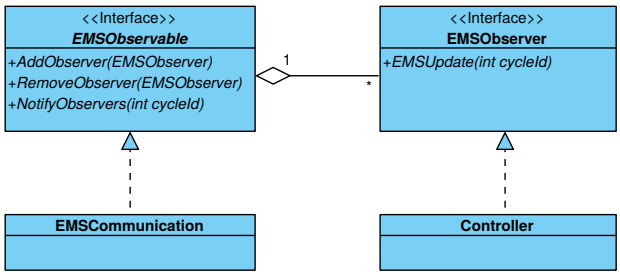


Abb. 5.18 Observer und Observable in der Steuerungsebene über den Ablauf eines Taktes des EMS (Eigene Darstellung)

Sollte zusätzlich ein Austausch von Objekten zwischen den Systemen über die Kommunikationskanäle notwendig werden, ist dies z.B. durch die *Boost*-Serialisierungs-Libraries möglich [49].

5.9.2 Entwurf der Kommunikation von Logging-Nachrichten

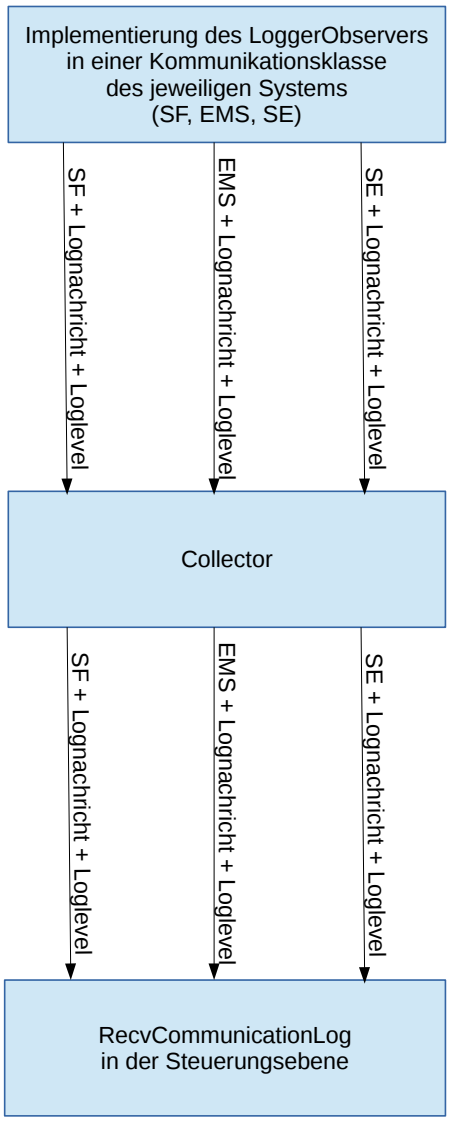


Abb. 5.19 Entwurf der Architektur der Kommunikation von Log-Nachrichten (Eigene Darstellung)

Gemäß den Anforderungen EMS-150 und SF-150 sollen im SF und im EMS jeweils Logs erstellt werden. Diese sollen z.B. in der Steuerungsebene abrufbar sein. Die Anforderungen EMS-180 und SF-180 sehen außerdem vor, dass die Logs als

Dateien bereitgestellt werden. Ebenfalls sollen gemäß EMS-160 und SF-140 Logs der Kommunikation erstellt werden. Während die Erstellung der Logs sowie die Bereitstellung der Logdateien mithilfe des in Abschnitt 5.14 beschriebenen Loggingkonzeptes sichergestellt wird, müssen diese Logs zusätzlich wie in Abschnitt 5.14.1 dargestellt an die Steuerungsebene versendet werden. Dazu sollen die dort beschriebenen *AllLogsObservables* und *AllLogsObserver* in der Steuerungsebene implementiert werden, um Logging-Nachrichten sämtlicher Systeme zu empfangen. Ebenfalls wird dort eine Klasse *RecvCommunicationLog* geplant, welche die Logs sämtlicher Systeme empfängt und das Observable informiert.

Zusätzlich wäre eine zentrale Sammelstelle hilfreich, die von allen Systemen Logging-Nachrichten empfängt, falls nötig zwischenspeichert und anschließend an die Klasse *RecvCommunicationLog* sendet. Das Zwischenspeichern von Nachrichten könnte etwa sinnvoll sein, wenn bereits Vorfälle geloggt wurden, bevor sich die Steuerungsebene mit dem jeweiligen System verbunden hat.

Um die Anforderungen zu erfüllen und die Steuerungsebene zu informieren bietet sich daher die in Abbildung 5.19 dargestellte Architektur an. Damit die Lognachrichten in der Steuerungsebene empfangen werden können, müssen diese zunächst versendet werden. Dazu kann eine Kommunikationsklasse des Systems jeweils den in Abschnitt 5.14 beschriebenen *LoggerObserver* implementieren und sich beim *LoggerObservable* des Systems registrieren. Anschließend können die Logs an den *Collector* gesendet werden, der als zentrale Sammelstelle dient und falls notwendig Logs zwischenspeichert. Die Klasse *RecvCommunicationLog* sollte sich beim Collector registrieren, damit keine weiteren Nachrichten zwischengespeichert werden. Anschließend sendet der Collector die gespeicherten sowie neu eingehenden Log-Nachrichten an die nun registrierte Klasse. In der Klasse *RecvCommunicationLog* werden diese demnach empfangen. Im Anschluss kann in der Steurungsebene das in Abschnitt 5.14.1 dargestellte *AllLogsObservable* über den Eingang des Logs informiert werden und seine Observer benachrichtigen. Dies ermöglicht etwa die Visualisierung der Log-Nachrichten in der GUI in Echtzeit.

Für das Versenden von Log-Nachrichten ist demnach eine unidirektionale Kommunikation ausreichend. Lediglich für die Registrierung der Systeme und der Steuerungsebene beim Collector wird eine bidirektionale Kommunikation benötigt. Aus den gleichen Gründen wie beim Broker sollte der Collector im EMS angesiedelt sein.

5.10 Grundlagen der Optimierung

In diesem Kapitel werden die Grundlagen der Optimierung geklärt. Wie in Abschnitt 5.7 beschrieben, werden die Samples in den jeweiligen Komponenten erstellt und die Optimierung kümmert sich um die Auswahl der Samples. Zunächst werden die Schnittstellen der Optimierung in Abschnitt 5.10.1 erläutert. Anschließend wird in Abschnitt 5.10.2 das Ziel der Optimierung geklärt. Zuletzt beschreibt Abschnitt 5.10.3 den Algorithmus der Optimierung.

5.10.1 Optimierungsschnittstellen

Die Eingabe für die Optimierung ist eine Liste von Samples pro Komponente, welche die möglichen Verläufe der Komponente für den Betrachtungszeitraum enthalten. Dazu kommen Rahmenbedingungen in Form der elektrischen und thermischen Grundlast und des Preises für den An- und Verkauf von Energie. Die Ausgabe ist der beste Plan, der von der Optimierung gefunden wurde. Darunter verstehen wir die Angabe welche Samples für jede Komponente verfolgt werden müssen. Zusätzlich werden Daten über den Verlauf der Optimierung in Logs und die Datenbank geschrieben.

5.10.2 Ziel der Optimierung

Ziel der Optimierung ist die Reduktion der Energiekosten eines Gebäudes durch geschickte Steuerung der EVS-Komponenten. Diese Steuerung erfolgt in einem 15 Minuten Rhythmus, in dem Parameter für Komponenten gesetzt werden, die Einfluss auf die Energiebilanz haben.
Grundlage für diese Steuerung ist die Verwendung von Samples (siehe Abschnitt 5.7). Diese sind pro Komponente verfügbar und liefern für jeden 15-Minutenschritt eine Energie- und Wärmebilanz. Damit sind die Art der verwendeten Komponenten sowie ihre Rahmenbedingungen für die Optimierungskomponente irrelevant, da nur mit ihren Samples gearbeitet wird und die Komponenten selbst darauf achten nur legale Samples zur Verfügung zu stellen.
Es lässt sich also das Optimierungsproblem $P = (S, f)$ definieren, wobei die Variablenmenge $X = \{x_1, ..., x_n\}$ die Menge der Komponenten und die dazu zugehörigen Samples darstellen. Hierzu existieren als Definitionsbereiche die Anzahl Samples, also die möglichen Indizes der jeweiligen Samples, $D_1, ..., D_n$. Die zu maximierende Zielfunktion f mit $f : D_1 \times ... \times D_n \rightarrow \mathbb{R}$ ordnet jedem so gewählten Sample eine Energie- und Wärmebilanz zu, welche dann mit einem An- und Verkaufspreis für Strom verrechnet wird, um einen Preis zu ermitteln. Die Menge S ist somit $S = \{s = \{(x_1, v_1), ..., (x_n, v_n)\} | v_i \in D_i$ und stellt alle wählbaren Pläne dar.

5.10.3 Optimierungsalgorithmus

Der von uns gesuchte Algorithmus soll eine möglichst optimale Lösung finden und muss in einer vorgegebenen Zeit das Ergebnis liefern. Bei der Zeiteinschränkung werden zwei Fälle unterschieden. Zum einen gibt es den Realzeitbetrieb, bei dem ein Optimierungsintervall von 15 Minuten zur Verfügung steht. Andererseits gibt es den Simulationsmodus mit einem Zeitraffer, bei dem das Optimierungsintervall ggf. deutlich kürzer ausfällt.
Es werden Heuristiken betrachtet, da diese in einer festgelegten Zeit ein möglichst

optimales Ergebnis liefern. Die Qualität des Ergebnisses einer Heuristik ist von der zur Verfügung stehenden Zeit abhängig. Dies hat zur Folge, dass die Simulation im Zeitraffer vom Realzeitbetrieb ggf. abweichen kann.

Auf lokaler Suche basierende Heuristiken finden bei uns keine Verwendung, da die verwendeten Samples zufällig generiert sind und es damit keinen Zusammenhang zwischen benachbarten Samples gibt. Dies führt zu einer globalen nicht linearen Optimierung. In einem solchen Umfeld eignen sich genetische Algorithmen und werden daher von uns verwendet (siehe Kapitel 3 Genetic Algorithms in [16]).

Bei uns sind Individuen Pläne. Sie enthalten also die Entscheidung welche der angebotenen Samples für jede Komponente gewählt werden. Anhand dieser Information können die Komponenten von der Optimierung gesteuert werden. Neue Individuen werden durch Rekombination erzeugt. Dabei werden von je zwei Exemplaren einer Generation zufällig Samples gewählt, sodass das neue Individuum genau ein Sample pro Komponente enthält. Zusätzlich werden zufällig einige der Werte mutiert, d.h. es wird ein Sample durch ein zufälliges anderes Sample der gleichen Komponente getauscht. Am Ende der Optimierung werden die Samples des besten Individuums verwendet, welches das Ziel der Optimierung besser erfüllt als alle anderen von der Optimierung betrachteten Individuen.

5.11 GUI-Konzept

Hier wollen wir zunächst auf das Mockup der GUI eingehen. Im Abschnitt 6.11 wird dann auf die Realisierung der GUI eingegangen.

Die Mockups wurden direkt mit dem QT-Designer erstellt. Der QT-Designer ist Bestandteil des QT-Frameworks [57] und kann zur Gestaltung von Benutzeroberflächen verwendet werden. Durch die Verwendung des Designers ist es durch wenige Anpassungen möglich, dass Mockup direkt in Source-Code zu übersetzen und dieses dann weiter zu verwenden.

Die GUI kann in zwei Teile aufgeteilt werden. Die Steuerungs-GUI kann zur Steuerung des Systems benutzt werden und die Evaluation-GUI zur Evaluierung des Systems. Diese Trennung wurde gewählt, da die Evaluation des Systems auch offline (ohne das EMS oder SF zu starten) durchgeführt werden kann und es so nicht erforderlich ist das System zu steuern.

5.11.1 Steuerungs-GUI

Die Steuerungs-GUI bietet die Möglichkeit das System zu steuern. Dies geschieht über die obere Steuerleiste (siehe Abbildung 5.20). Im linken Teil der Leiste kann das System, wie in Anforderung SE-EMS-05 gefordert, gestartet, gestoppt und konfiguriert werden. Die Konfiguration ist über zwei Wizards möglich. Der Scenario-Wizard wird genutzt um neue Scenarios an zu legen (Anforderung SE-60). Beim Anlegen

des Scenarios kann ein Gerätepool erstellt werden (Anforderung SE-50). Im Run-Wizard können Einstellungen zum aktuellen Lauf vorgenommen werden. Hier kann z.B. der in Anforderung SE-10 geforderte Zeitraffer eingestellt oder Wetterdaten (Anforderung SE-70) sowie ein Grundlastmodell (Anforderung SE-80) mitgegeben werden.

Der rechte Teil der Leiste ermöglicht es die Loginformationen ein zu stellen. Hier ist es möglich nur Loginformationen zu bestimmten Systemen anzeigen zu lassen.

Im mittleren Teil der GUI werden die Werte der Komponenten in Tabellen angezeigt. Wie in Anforderung SE-EMS-20 gefordert, ist es so möglich den Zustand der einzelnen Komponenten während der Ausführung des Systems zu verfolgen.

Im unteren Teil gibt es ein Fenster wo die gefilterten Logginginformationen angezeigt werden. Dieses Fenster dient unter Anderem zum Anzeigen des von in Anforderung SE-EMS-10A geforderten Optimierungszustand.

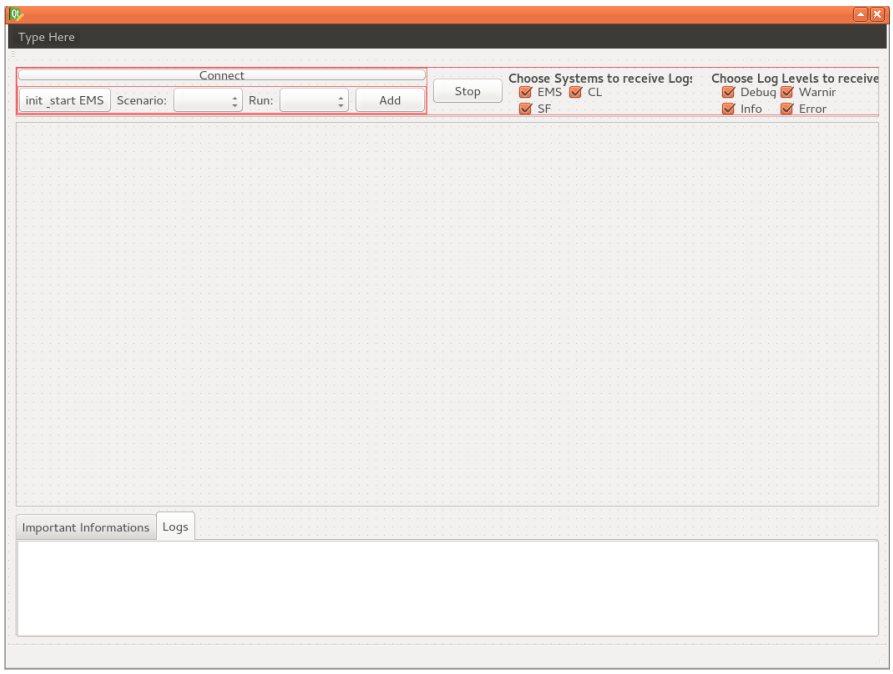


Abb. 5.20 Steuerungs-GUI Mockup

5.11.2 *Evaluation-GUI*

In der Evaluation-GUI können drei verschiedene Ansichten geschaltet werden (siehe Abbildung 5.21). Die GUI wurde in diese drei Ansichten aufgeteilt, da so eine übersichtliche Darstellung der Daten möglich ist. Besonders die Darstellung der Graphen profitiert von dem so gewonnen Platz. Über den ersten Reiter *Selection* können Kenngrößen ausgewählt werden, die anschließend gegenübergestellt werden sollen. In den anderen beiden Ansichten (Trend Chart und Table View) werden die ausgewählten Daten dann jeweils als Diagramm und als Tabelle dargestellt.

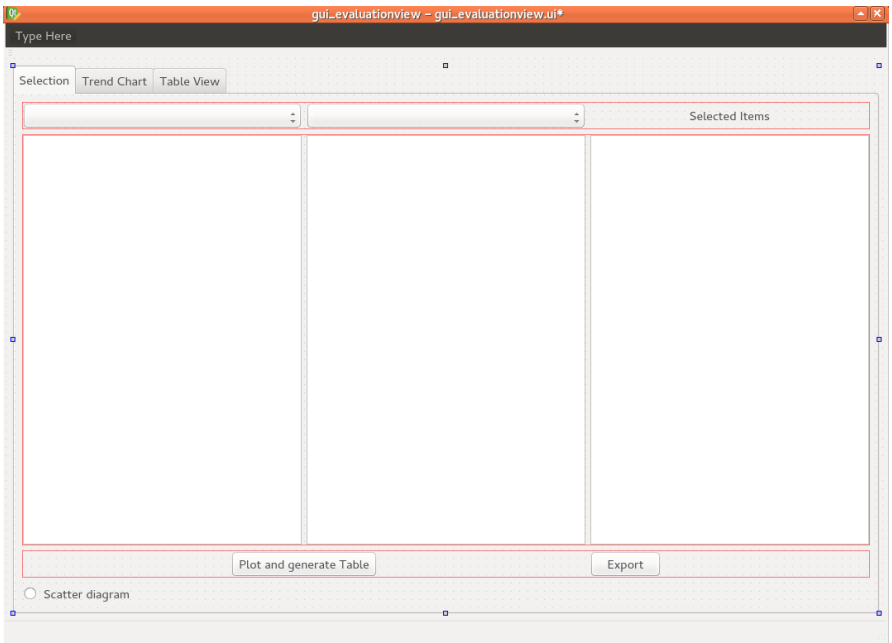


Abb. 5.21 Evaluation-View Mockup

5.12 Zeitgeber

In diesem Abschnitt wird der Zeitgeber als zentrale Komponente für die Regelung des Systemablaufs entworfen. Die Anforderung EMS-70 besagt, dass das EMS die EVS-Komponenten im vom Systemtakt vorgegebenen Abstand aktualisieren muss. So soll z.B. in jedem Taktschritt ein neues Sampling der EVS-Komponenten durchgeführt werden. Daher muss es eine Komponente geben, die den Systemablauf steuert.

Die Steuerung der EVS-Komponenten erfolgt in einem 15-Minuten Takt. Dabei werden die Steuerungsparameter von einer Optimierungs-Komponente ermittelt (vgl. Abschnitt 5.10.2). Damit diese Parameter korrekt berechnet werden können, muss ein bestimmter Systemablauf eingehalten werden.

Wie bereits festgestellt ist die Reihenfolge, in der die Komponenten des Systems ihre Aufgaben ausführen, sehr wichtig für die Korrektheit für das Ermitteln von sinnvollen Steuerparametern. Je nach Szenario und den EVS-Komponenten, die gesteuert werden sollen, variiert die Anzahl der Systemkomponenten und die Reihenfolge, so dass eine komplett feste Reihenfolge nicht möglich ist. Damit ein Zeitgeber trotzdem den Systemablauf sinnvoll steuern kann, muss den Komponenten eine Priorität zugeordnet werden können, mittels der die Reihenfolge festgelegt werden kann.

Neben der Reihenfolge muss ein Zeitgeber auch alle Systemkomponenten ansteuern können. Da diese Komponenten beliebig erweitert bzw. neue hinzukommen können ist die Verwendung der verschiedenen Schnittstellen der einzelnen Komponenten nicht sinnvoll. Dementsprechend muss der Zeitgeber eine Schnittstelle vorgeben, die von Komponenten, die von dem Zeitgeber gesteuert werden sollen, verwendet werden muss.

Das Observer-Pattern erlaubt das Beobachten von Objekten. Dabei müssen Observer (Objekte die eine Observer-Schnittstelle implementieren) sich bei den „Beobachteten“ anmelden. Bei Änderungen des Zustands werden die „update“-Funktionen der Beobachter ausgeführt. [15, S. 293ff.]

Grundsätzlich lässt sich der Großteil der Funktionalität des Zeitgebers auf dieses Design-Pattern abbilden. Der Zeitgeber wäre dabei der „Beobachtete“, während die Systemkomponenten, deren Ablauf gesteuert wird, Observer wären. Für die Ausführung eines Taktzyklus wäre ein Aufruf der „notify“-Funktion des Beobachteten notwendig. Das Observer-Pattern behandelt alle Observer mit gleicher Priorität, dementsprechend muss das Pattern um Prioritäten erweitert werden, damit eine Reihenfolge festgelgt werden kann, in der die Observer benachrichtigt werden. Die geplante Architektur ist in Abbildung 5.22 dargestellt, wobei „ClockPriority“ die Priorität der Observer widerspiegelt. Daneben werden Methoden für die An- und Abmeldung der Observer beim Observable sowie zum Starten und Beenden des Zeitgebers benötigt. Eine Implementierung des Observables als Singleton stellt sicher, dass nur ein Taktgeber pro System aktiv ist. Der Controller des Systems könnte jeweils die „ClockObserver“-Schnittstelle implementieren und sich beim Zeitgeber registrieren. Anschließend könnte dort die weitere Ausführung des Taktes veranlasst werden, z.B. durch Benachrichtigung der Feldkommunikation.

Der Zeitgeber ist aber auch insbesondere für die Zeit zuständig. Er muss dafür sorgen, dass ein Zyklus immer genau der festgelegten Dauer entspricht (15 Minuten im Realzeitbetrieb oder kürzer bei Simulation im Zeitraffer).

Proxy-Zeitgeber im Simulationsframework

Laut SF-60 muss zudem das SF den EVS-Komponenten Simulationsdaten zu dem im EMS aktuellen Simulationsschritt liefern, da das SF das Szenario initialisiert.

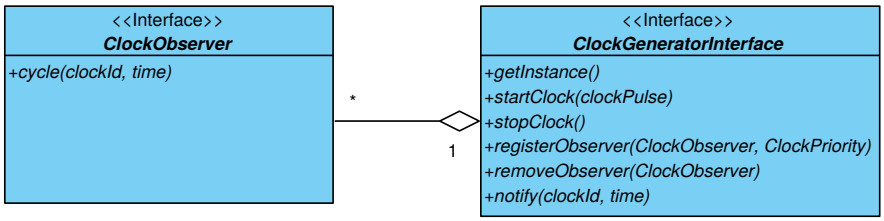


Abb. 5.22 Entwurf der Architektur des Zeitgebers (Eigene Darstellung)

Außerdem muss beim Starten des EMS die Simulation implizit starten (vgl. SF-70A).

Um diese Anforderungen zu erfüllen, ist geplant, dass der Zeitgeber (ClockGenerator) des EMS einen Proxy-Zeitgeber im SF synchron hält. Da im Gegensatz zum ursprünglichen Proxy-Pattern (vgl. [15, S. 207ff.]) direkte Funktionsaufrufe nicht möglich sind weil sich die Zeitgeber in unterschiedlichen Systemen befinden, ist der Austausch von Informationen über die Systemkommunikation (vgl. Kapitel 5.9.1) angedacht. Dies kann mithilfe der in Abbildung 5.23 dargestellten Architektur realisiert werden. Eine „ProxyClock“ registriert sich beim Zeitgeber des EMS als Observer und informiert wiederum über die Systemkommunikation einen „ProxyClockGenerator“ im SF. Bei diesem ProxyClockGenerator kann sich wiederum der Controller des Simulationsframeworks registrieren. Dabei muss bei der Realisierung beachtet werden, dass das EMS auch ohne SF lauffähig sein soll, sodass die Ausführung eines Taktes in der ProxyClock nicht zum Blockieren führen darf, wenn das Simulationsframework nicht erreichbar ist.

Da weiterhin die optionale Anforderung besteht, dass das Simulationsframework die Simulation ohne das EMS starten können soll (vgl. SF-80A), sollte der Proxy-Zeitgeber im SF durch einen anderen Zeitgeber austauschbar sein. Dies kann durch die Verwendung der „ClockGenerator“- und „ClockObserver“-Interfaces sichergestellt werden.

5.13 Datenbankkonzept

Das Datenbankkonzept ist eine zentrale Aufgabe des Projekts. Die Datenbank (DB) ist innerhalb der Architektur nicht nur zum Speichern der Daten zuständig, sondern auch für deren Austausch unter den Systemen. Näheres zur Einbettung der DB in der Architektur und dessen Aufgabe ist im Abschnitt 5.13.1 erläutert.

Für die jeweiligen Systeme sind zu bestimmten Zeitpunkten entsprechende Datenfelder zum Speichern und Lesen in der DB notwendig. So benötigt das EMS die Grundlastprognosedaten (Anf. EMS-30A) und Informationen über angeschlossene EVS-Komponenten des Gebäudes (Anf. EMS-10A) zur Optimierung. Das Simulationsframework benötigt neben den Grundlastprognosedaten und Wetterprognose-

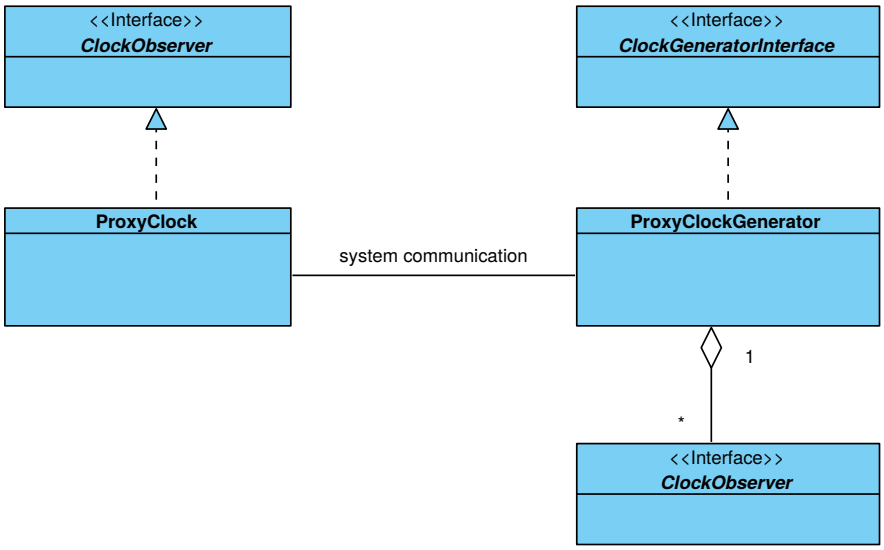


Abb. 5.23 Entwurf des ProxyClock-Generators

daten, zu deren Verrauschung (Anf. SF-40A, SF-55), ebenfalls die Kenndaten der EVS-Komponenten um diese zu Initialisieren (Anf. SF-60). Die Steuerungsebene hingegen soll den Zustand der EVS-Komponenten des Gebäudes (Anf. SE-EMS-20) und der Optimierung (Anf. SE-EMS-10A) darstellen und braucht für diese Aufgabe deren Laufdaten.

Genaueres zur Datenstruktur der Datenbank ist in Abschnitt 5.13.4 festgehalten. Um die anfallenden Daten schnell und zuverlässig abzuspeichern, haben wir uns für den Einsatz eines Datenbankmanagementsystems (Datenbankmanagementsystem (DBMS)) entschieden. Grundlage für den Entwurf der DB war das Datenmodell, das in Abschnitt 5.3 bereits beschrieben wurde.

Wir setzen in der Entwicklung MySQL[45] ein. Ein Grund war, dass diese Software unter *GNU General Public License* [11] steht, sodass wir die Software in unserem Projekt einsetzen dürfen. Problematisch ist diese Lizenzierung, wenn das Projekt später als kommerzielle bzw. proprietäre Software vertrieben wird, da die GPL-Lizenz nur eine Veröffentlichung der Software unter der GPL-Lizenz vorsieht, sowie eine Veröffentlichung des Quelltextes [22] [56] [44] [12]. Möglichkeiten dies zu beheben wäre der Kauf einer kommerziellen Lizenz für MySQL oder der Einsatz der Datenbankengine MariaDB [37]. Diese wurde unter der *Lesser General Public License* (LGPL) [14] veröffentlicht. Bei dieser Lizenz müssen nur die Teile veröffentlicht werden, die LGPL-Software verändern [4]. Bei unserem Projekt nutzen wir jedoch nur die Funktionen der zur Verfügung gestellten Programmbibliothek, ohne Änderungen daran vorzunehmen. Unsere Arbeiten sind nach der LGPL kein abgeleitetes Werk mehr und dürfen auch unter einer proprietären Lizenz weitergegeben werden.

Ein MySQL-Server kann mehrere relationale Datenbanken verwalten, in denen wiederum mehrere Tabellen angelegt werden können. Auf den MySQL-Server können etliche Clients gleichzeitig zugreifen und Anfragen absetzen. Diese Eigenschaft nutzen wir, da es einen zentralen MySQL-Server gibt, auf den die einzelnen Lösungen Visualisierungskomponente, Energiemanagementsystem und Simulationsframework zugreifen.

5.13.1 Architektur der Datenbank

Bei der Datenbankarchitektur haben wir uns an der ANSI-SPARC-Architektur oder auch Drei-Ebenen-Architektur orientiert [35]. Die Architektur wird dabei in drei Ebenen aufgeteilt. Die erste Ebene ist das Frontend, welches bei uns die Visualisierungskomponente ist. Hier kann der Benutzer die Szenarien anlegen, speichern und sich die Evaluation anschauen. Die zweite Ebene ist die konzeptionelle Ebene. Hier wird festgelegt, welche Daten gespeichert werden und wie ihre Beziehungen zueinander sind. Dies haben wir mittels EER-Diagrammen visualisiert. Die dritte und letzte Ebene ist die interne Ebene, bei der die genaue Speicherung der Daten auf dem Speichermedium angegeben wird.

5.13.2 Externe Datenbankebene

Das Frontend für den Benutzer ist bei uns die Visualisierungskomponente. Diese Komponente wird in den Kapiteln 5.11 und 6.11 genauer vorgestellt. In diesem Abschnitt wird deshalb auf die Schnittstelle zwischen Frontend und Backend genauer eingegangen. Dafür haben wir die Klasse Dataaccesslayer eingeführt, die aus einem Methodenaufruf aus dem Programm heraus dafür sorgt, dass Daten in die Datenbank gespeichert oder ausgelesen werden. Zum Einen übersetzt diese Klasse die C++-Objekte in Daten, die in der Datenbank gespeichert werden können. Zum Anderen werden die Abfragen so aufgearbeitet, dass die Ergebnisse in C++-Objekten präsentiert werden.

Der Vorteil solch eines Data-Access-Layers liegt vor allem an der Abstraktion von den tatsächlich vorhandenen Tabellen in der Datenbank. Die Programmierer, die auf die Methoden im Data-Access-Layer zugreifen, benötigen keine Kenntnisse über die zugrunde liegende Datenbank oder über SQL-Befehle. Sie benötigen lediglich eine Schnittstellenbeschreibung, welche Objekte sie den Methoden übergeben müssen, bzw. welche Art von Objekten sie bei einer Abfrage zurückerhalten. Der Data-Access-Layer kümmert sich in der Folge darum, dass die zurückgegebenen Objekte korrekt initialisiert worden sind und Attribute korrekt gesetzt sind.

Lediglich die Programmierer des Data-Access-Layer benötigen das Wissen, wie die Daten mittels SQL-Statements in die Datenbank gespeichert und abgefragt werden. Ebenfalls von Vorteil ist es, dass alle Methoden, die SQL-Befehle ausführen,

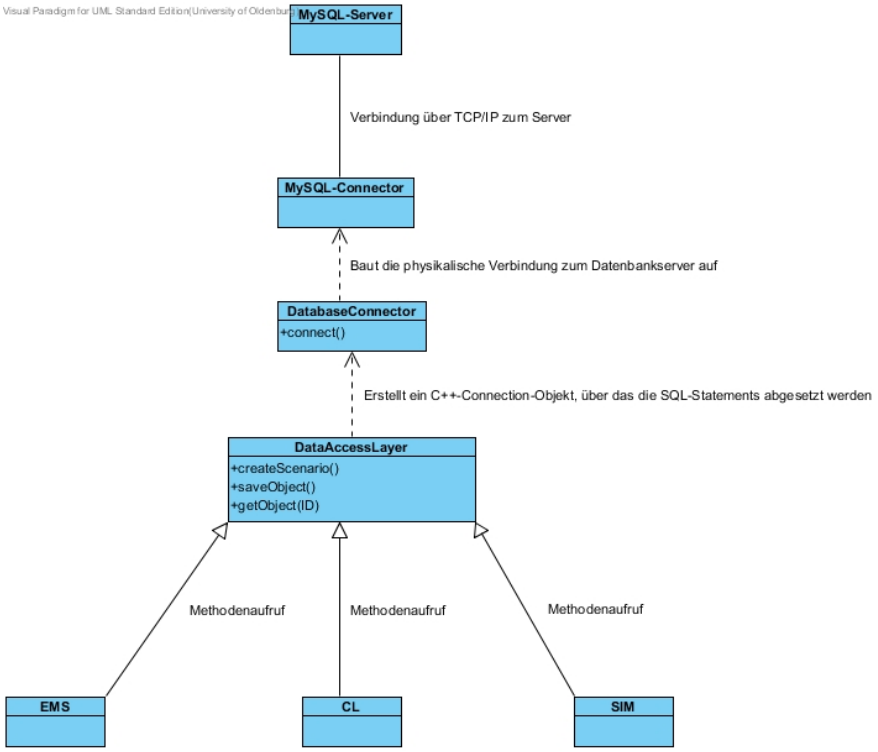


Abb. 5.24 Die Grafik zeigt, wie die Klasse DataAccessLayer in die Architektur eingebunden ist. Die Optimierungskomponente (EMS), das Simulationsframework (SIM) und die Visualisierungskomponente (CL) rufen die Methoden im DataAccessLayer auf. Dieser bringt die Objekte in ein für die Datenbank benötigtes Format, baut die Verbindung zum Datenbankserver auf und speichert die Daten.

in einer Klasse vereint sind. Dadurch wird die Wartung an der Software erleichtert. Sind Fehler vorhanden oder hat sich das Datenbankdesign geändert, müssen nur in einer einzigen Klasse Veränderungen vorgenommen werden. Da der Data-Access-Layer in allen Bereichen der Software benutzt wird, sind nach einer Fehlerkorrektur automatisch alle Softwarebestandteile auf dem aktuellen Stand.

Der Einsatz des Data-Access-Layer bietet weiterhin den Vorteil, dass das Datenbankmanagementsystem sehr schnell ausgetauscht werden kann. Durch die Aufteilung wie in Abbildung 5.24 gezeigt, wird die Verbindung durch die Klasse DatabaseConnector aufgebaut. Wird hier die Verbindung zu einer anderen Datenbank eingetragen, kann jede Datenbank verwendet werden, die einen C++-Treiber zur Verfügung stellt und mittels SQL-Syntax angesprochen wird.

5.13.3 Interne Datenbankebene

Die interne Datenbankebene beschreibt, wie die Daten genau gespeichert werden. Wie bereits in Abschnitt 5.13 angesprochen wurde, haben wir uns für MySQL als Datenbankmanagementsystem entschieden. Als Speicherengine verwenden wir InnoDB, welches in der von uns verwendeten MySQL-Version 5.6 das Standardspeichersystem ist. InnoDB bietet die Vorteile, dass sowohl Transaktionssicherheit als auch referenzielle Integrität sichergestellt wird. Für uns ist vor allem die referenzielle Integrität wichtig, da viel mit Fremdschlüsseln in der Datenbank gearbeitet wird. Durch InnoDB wird sichergestellt, dass Fremdschlüssel nur auf existierende Datensätze verweisen. Es ist dadurch leichter die Datensätze in der Datenbank konsistent, das bedeutet in diesem Fall ohne „Karteileichen“, zu halten.

Auf dem Datenbankserver werden verschiedene Datenbanken angelegt, um alle Daten zu speichern. Auf das genaue Design der einzelnen Datenbanken und der Tabellen wird in dem nächsten Abschnitt eingegangen. Es gibt zum Einen die Konfigurationsdatenbank und zum Anderen eine Datenbank für jedes angelegte Szenario. Die Konfigurationsdatenbank wird automatisch beim ersten Start des Systems angelegt. Neben einem vorkonfigurierten Szenario werden dort alle EVS-Komponenten und die bereits angelegten Szenarien verwaltet. Die vom Benutzer angelegten Szenarien werden in extra angelegten Datenbanken auf dem gleichen Server verwaltet. Dort werden die genauen Parameter der Komponenten, die Wetterdaten und die tatsächlich durch die Komponenten produzierten Werte gespeichert. Die Aufteilung in die beiden Datenbanktypen wurde getroffen, da EVS-Komponenten Wertebereiche für die Parameter annehmen können, der in der Konfigurations-Datenbank gespeichert ist. In einem fest angelegten Szenario können die Komponenten nur noch einzelne Werte als Parameter haben. Zum Beispiel ist die maximale Leistung eines Erzeugers mit Start des Szenarios festgelegt.

Durch die Aufteilung konnte des Weiteren der Wartungsaufwand minimiert werden. Während der Entwicklung wurde das Datenbankdesign oft verändert, erweitert und an neue Anforderungen angepasst. Durch die Trennung der einzelnen Szenarien konnten alte Datenbanken mit alten Läufen zu Testzwecken auf dem Server belassen werden. Bei einer Änderung der Tabellen reichte es, ein neues Szenario anzulegen. Dadurch wurde auf dem Datenbank-Server automatisch eine neue Datenbank mit den neuen Tabellen angelegt. Die alte Datenbank konnte dennoch über Tools ausgelesen werden. Von Vorteil war die Aufteilung in mehrere einzelne Datenbanken auch deshalb, weil einzelne Datenbank-Dumps gesichert und wieder herstellt werden konnten. So konnten die gleichen Läufe auf verschiedenen Rechnern untersucht und evaluiert werden.

5.13.4 Konzeptionelle Datenbankebene

Die Datenbankschema wurden zweckgemäß und insbesondere in Hinblick auf Erweiterbarkeit des Pools von EVS-Komponenten (Anf. SE-50), weiteren Input-Werten und Output-Werten erstellt.

Wie in Abschnitt 5.13.3 erklärt wurde, sind in dem Projekt zwei Datenbanken entwickelt worden. In der Konfigurationsdatenbank werden dabei die Szenarien, EVS-Komponenten und Input-Daten gespeichert. Jedes vorhandene Szenario in der Konfigurationsdatenbank besitzt dabei ein entsprechendes Pendant als Szenario-Datenbank. Die Szenarios in der Konfigurationsdatenbank dienen also nur der Auswahl des aktuellen Szenarios. Sie enthält außerdem alle bisher erstellten EVS-Komponenten und Input-Daten, in unserem Fall Wetterdaten (Anf. SE-70) und Grundlastdaten (Anf. SE-80) zur Konfiguration eines neuen Szenarios.

Zu den EVS-Komponenten und Input-Daten können jederzeit weitere Hinzugefügt werden ohne das Änderungen an der Datenbank notwendig wären. In Abbildung 5.25 ist das Datenbankschema der Konfigurationsdatenbank zu sehen.

Eine besondere Rolle haben die Semantiken. Jedes Attribut, egal ob einer EVS-Komponente, eines Inputs oder eines Outputs, wird einer bestimmten Semantik zugeordnet. Über die Semantik ist ein eindeutiges Mapping von Input- und Output-Werten auf bestimmte EVS-Komponenten-Attribute gewährleistet. Eine genaue Beschreibung der Tabellen und Attribute der Konfigurationsdatenbank sind im Anhang Tabelle A.1 zu finden.

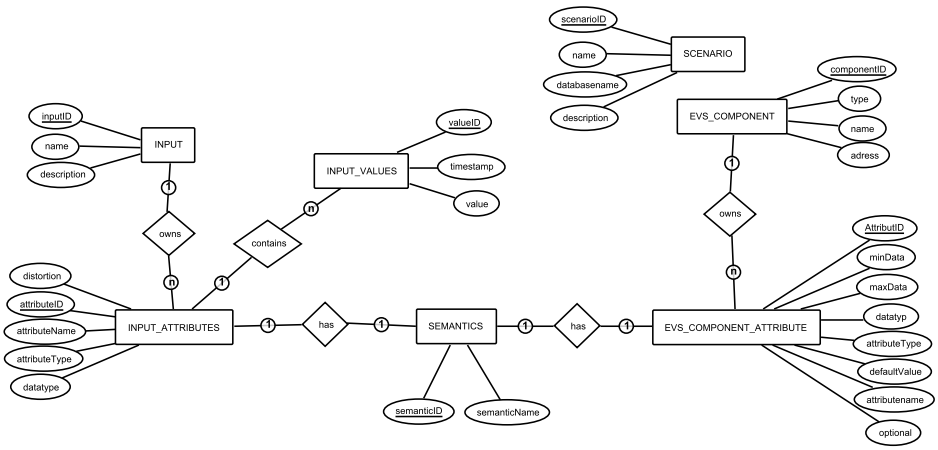


Abb. 5.25 Datenbankschema der Konfigurationsdatenbank.

Die Szenario-Datenbank enthält jeweils alle Daten eines Szenarios. Für jedes Szenario wird eine Datenbank mit dem selben Datenbankschema erstellt. Dadurch lässt sich das Szenario leicht exportieren. Abbildung 5.26 zeigt das Datenbankschema der

Szenario-Datenbank in vereinfachter Form ohne Attribute. Das gesamte EER der Szenario-Datenbank (Abbildung A.3), sowie eine Tabelle A.3 mit Beschreibungen ist im Anhang einsehbar.

Das Datenbankschema ist zum einfacheren Verständnis in Baumstruktur dargestellt, wobei das Szenario die Wurzel und alle weiteren Tabellen die Blätter verbildlichen. Als Ausnahme in der Struktur ist die bereits erwähnte Semantik zu betrachten, auf Grund der bereits beschriebenen besonderen Rolle. Der linke Zweig enthält Daten vom Szenario, die dieses selbst festlegen und definieren. Dazu gehören das Gebäude, die EVS-Komponenten und die Input-Daten.

Der rechte Zweig, beginnend den Einstellungen eines Laufs in der Tabelle Run, enthält alle Werte die während eines Szenariodurchlaufs von den simulierten EVS-Komponenten (*Component_Rundata*), dem Simulationsframework (Zweig ab textit-Sim_Run) und der Optimierung des EMS (Zweig ab *EMS_Run*) geschrieben werden. Diese Werte sind später vom Nutzer abrufbar und können in der GUI angezeigt werden. Weitere Ausgabewerte können mit der vorhandenen Struktur einfach in das Gesamtsystem eingefügt werden. Zu Beachten ist immer, dass eine entsprechende Semantik ebenfalls eingefügt werden muss, sofern sie noch nicht vorhanden ist.

Während der linke Zweig eines Szenarios statisch bleibt, können im rechten Zweig eine unbestimmte Anzahl an Läufen hinzukommen. Werden neue Input-Daten oder EVS-Komponenten für ein Szenario in Betracht gezogen, muss hierfür ein neues Szenario erstellt werden.

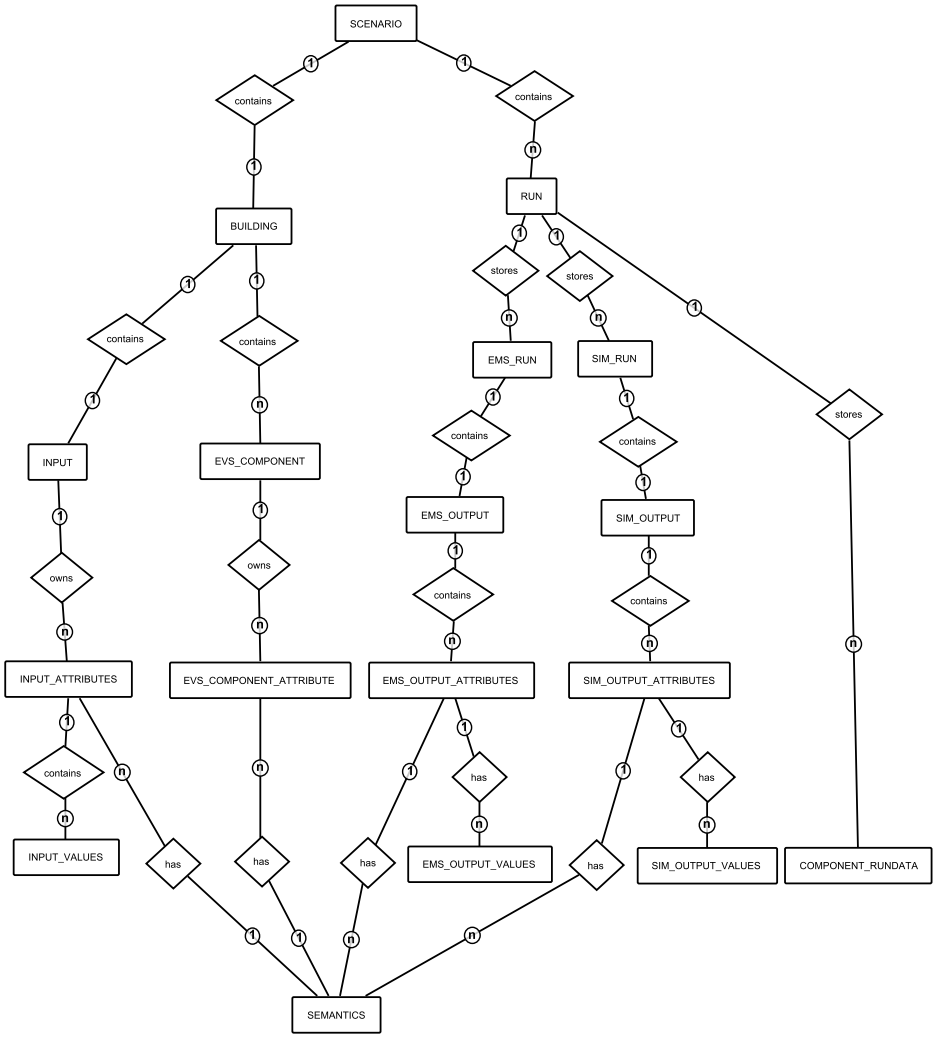


Abb. 5.26 Vereinfachtes Datenbankschema der Szenario-Datenbank ohne Attribute und in Baumstruktur.

5.14 Loggingkonzept

Sowohl für den Administrator als auch den Entwickler ist es notwendig, Fehler oder unerwünschtes Verhalten schnell zu erkennen bzw. zu korrigieren. Damit dies möglich ist, muss der Zustand der Systeme einsehbar sein. Deshalb haben wir uns entschieden einen Logger zu verwenden, der wichtige Informationen speichert und für den Benutzer einsehbar ist. Die gespeicherten Informationen können sehr unterschiedlicher Natur sein. Sie können Fehlernachrichten, Debuginformationen, Variablen der Optimierung etc. darstellen.

Das hier vorgestellte Konzept legt sich dabei nicht auf einen bestimmten Logger fest, sondern beschreibt zunächst die grundlegende Struktur des Loggers im System, d.h. wie er ins System integriert ist und wie geloggte Nachrichten transportiert werden. Die Realisierung des Loggers befindet sich im Kapitel Realisierung in Abschnitt 6.5. Um von der späteren Implementierung des Loggers zu abstrahieren, gehen wir zunächst von einer Klasse Logger aus, welche einen solchen realisiert. Ein Logger, ist dann ein Objekt, welches Nachrichten entgegennehmen und verarbeiten kann, um z.B. eine Logdatei anzulegen oder um Nachrichten mit zusätzlichen Informationen, wie z.B. Funktionsname oder Datum, anzureichern.

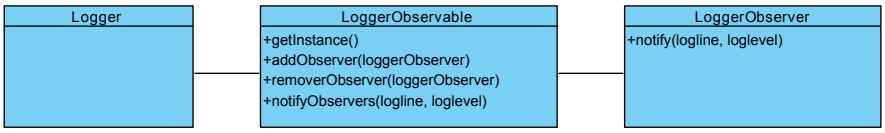


Abb. 5.27 Grundlegende Architektur des Loggingskonzepts

Das Grundkonzept beinhaltet neben diesem Logger ein Observerpattern. (Siehe Abbildung 5.27) Wenn ein Logger eine Nachricht empfängt und verarbeitet, sendet er diese Nachricht an das LoggerObservable des Systems. Dieses Observable ist ein Singleton und somit wird gewährleistet, dass es pro System nur ein LoggerObservable existiert. Dieses LoggerObservable ist dabei vergleichbar mit einer Post. Es ist die zentrale Anlaufstelle zum Entgegennehmen der Nachrichten von ein oder mehreren Loggern und zum Weiterleiten dieser Nachrichten an ein oder mehreren Observern. Dazu registrieren sich LoggerObserver beim LoggerObservable und werden dann über eingehende Logs informiert. Diese Nachrichten können vorher von einem Logger mit zusätzlichen Informationen angereichert werden.

Der Vorteil dieser Architektur ist, dass es um neue Observer erweiterbar ist und somit alle Stellen informiert werden können, welche die Nachrichten benötigen. Gleichzeitig wird die mögliche Implementierung der Logger kaum eingeschränkt und kann somit sehr flexibel an Bedürfnisse des realisierten Systems angepasst werden. Dies lässt sich für unsere Software nutzen, indem z.B. ein Observer implementiert werden kann, der die Steuerungsebene über erstellte Log-Nachrichten informiert. Dies

ermöglicht den Benutzer die Nachrichten der Systeme gebündelt in der Steuerungsebene abzurufen.

5.14.1 Verteilung der Lognachrichten an die Steuerungsebene

Die in Abschnitt 5.14 beschriebene Architektur zeigt auf, wie die Logs pro System abrufbar gestaltet werden können. Für die Steuerungsebene ist aber zusätzlich relevant, dass die Nachrichten aller Systeme gebündelt und möglichst unverzüglich abrufbar sind. Dazu kann zunächst der in Abschnitt 5.9.2 beschriebene Collector in Verbindung mit dem in Abbildung 5.27 dargestellten Observer dienen: Eine Klasse des Systems, welche für die Systemkommunikation zuständig ist, kann sich beim LoggerObservable als Observer anmelden. Die Implementierung des Observers in der jeweiligen Kommunikationsklasse sollte dafür sorgen, dass die Log-Nachrichten an den Collector im EMS weitergeleitet werden.

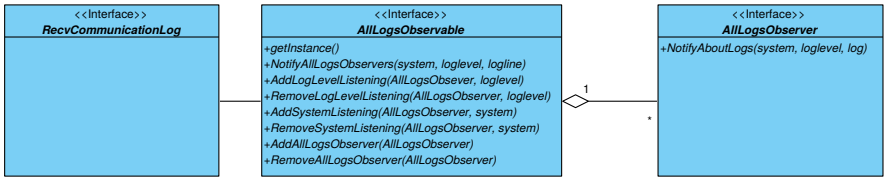


Abb. 5.28 Architektur der Verteilung der Lognachrichten an die Steuerungsebene

Zusätzlich ist es erforderlich, dass die Nachrichten aller Systeme in der Steuerungsebene auch empfangen werden. Dazu kann die in Abbildung 5.28 dargestellte Architektur dienen.

Zunächst ist eine Klasse RecvCommunicationLog erforderlich, die beim Collector die Log-Nachrichten abruft. Wird nun ein Log in der Klasse RecvCommunicationLog empfangen, kann diese das AllLogsObservable informieren. Die erneute Verwendung eines Observerpatterns stellt sicher, dass die Benutzeroberfläche der Steuerungsebene umgehend aktualisiert werden kann, wenn eine Nachricht eintrifft. Im Gegensatz zu dem in Abschnitt 5.14 dargestellten Observable wird die Information ergänzt, welches System die Nachricht versendet hat. Somit kann das AllLogsObservable Methoden anbieten, um die Logs etwa nach speziellen Loglevels (z.B. nur Fehlermeldungen) oder nach speziellen Ursprungssystemen (z.B. nur Logs des EMS) zu filtern. Für die Klasse AllLogsObservable ist ebenfalls die Verwendung des Singleton-Patterns vorgesehen, damit in der Steuerungsebene nur ein AllLogsObservable existiert, welches die Nachrichten aller Systeme empfängt. Der Controller der GUI kann sich nun als Observer beim AllLogsObservable registrieren und wird auf diesem Wege umgehend über alle aktuellen Logs informiert.

5.15 Grundlastberechnung nach VDI4655

Eine wichtige Komponente des Systems ist die Berechnung der elektrischen und thermischen Grundlast des simulierten Gebäudes. Diese Berechnung geschieht nach der Richtlinie VDI4655 [59]. Diese bietet Referenzlastprofile von Ein- und Mehrfamilienhäusern für den Einsatz von Heizungsanlagen mit Kraft-Wärme-Kopplung (KWK-Anlagen) und dient primär als Instrument zur Dimensionierung der KWK-Anlagen in den entsprechenden Gebäudetypen und ihrer Wirtschaftlichkeitsberechnung. Im Rahmen des Projekts wird die Richtlinie genutzt um für das simulierte Gebäude den viertelstündlichen Bedarf an elektrischer und thermischer Energie zu berechnen. Bei dem simulierten Gebäude handelt es sich für diese Berechnung um ein Mehrfamilienhaus, dessen Energiebedarf durch die Eingabe von jährlichen Gesamtbedarf an thermischer und elektrischer Energie durch den Benutzer skaliert werden kann.

5.15.1 Anwendung der Richtlinie

Zur Anwendung der Richtlinie sind verschiedene Arbeitsschritte nötig, die im Weiteren erläutert werden. Zunächst müssen die Input-Parameter für die Berechnung gesetzt werden.

Folgende Eingabeparameter sind vom Anwender zu setzen:

- Simulationszeitraum
- Wetterdaten
- Jahresbedarf an thermischer und elektrischer Energie

Folgende Parameter werden für das System festgelegt:

- Kenndaten des Gebäudes
- Gebäudestandort

Mit Hilfe des Simulationszeitraums und der zugehörigen Wetterdaten wird zunächst für jeden Tag der Typtag festgelegt, welcher den Energiebedarf des Gebäudes für die jeweilige Typtagkategorie repräsentiert. Diese Typtagkategorie ist abhängig von der Jahreszeit (Sommer-, Winter- oder Übergangstag), dem Kriterium Werk- oder Sonntag und dem Kriterium heiter oder bewölkt. Die nachfolgende Tabelle zeigt die zehn verschiedenen Kategorien.

Jahreszeit	Werktag W		Sonntag S	
	Heiter H	Bewölkt B	Heiter H	Bewölkt B
Übergang Ü	**ÜWH**	**ÜWB**	**ÜSH**	**ÜSB**
Sommer S	**SWX**		**SSX**	
Winter W	**WWH**	**WWB**	**WSH**	**WSB**

Nach diesen Kategorien sind die Tageslastgänge für den Bedarf an elektrischer und thermischer Energie unterteilt. Die Kategorisierung erfolgt für jeden Tag, der sich innerhalb des Simulationszeitraums befindet. Das Kriterium Jahreszeit ist abhängig von der Tagesmitteltemperatur eines Tages, die mittels der übergebenen Wetterdaten berechnet werden kann.

- Definition Wintertag: Tagesmitteltemperatur unter 5 °C
- Definition Übergangstag: Tagesmitteltemperatur zwischen 5 °C und 15 °C
- Definition Sommertag: Tagesmitteltemperatur über 15 °C

Für die Einteilung nach Werktagen und Sonntagen wird die kalendarische Zuordnung aus dem Simulationszeitraum verwendet. Sonntage sind nach der hier verwendeten Definition auch alle gesetzlichen Feiertage. Der Bedeckungsgrad eines Tages wird ebenfalls mit den vorhandenen Wetterdaten bestimmt. Dieser wird in Achteln gemessen. Zur Unterscheidung von heiteren und bewölkten Tagen wird laut der Richtlinie definiert:

- Ein Tag ist heiter, wenn das Tagesmittel des Bedeckungsgrades < 5/8 ist.
- Ein Tag ist bewölkt, wenn das Tagesmittel des Bedeckungsgrades > 5/8 ist.

Für Sommertage ist gemäß der Richtlinie keine Unterscheidung zwischen heiteren und bewölkten Tagen notwendig. Ein weiterer Einflussfaktor ist der Standort des simulierten Gebäudes, da die klimatischen Bedingungen in Deutschland unterschiedlich sind. Der Deutscher Wetterdienst (DWD) unterteilt Deutschland in 15 (TRY01 bis TRY15) verschiedene Klimazonen. Vom DWD erstellte Tabellen für die jeweiligen Klimazonen (siehe Abbildung 5.29) liefern Faktoren, mit denen für den bestimmten Typtag zusammen mit dem jährlichen Bedarf des Gebäudes an elektrischer und thermischer Energie der Tagesbedarf berechnet werden kann. Der Standort des simulierten Gebäudes ist für das im Rahmen des Projekts entwickelte System festgelegt auf die Klimazone, in der sich die Stadt Oldenburg befindet.

TRY03

EFH	ÜWH	ÜWB	ÜSH	ÜSB	SWX	SSX	WWH	WWB	WSH	WSB
$F_{Heiz,TT}$	1,7169E-03	2,4087E-03	2,4827E-03	2,2563E-03	0,0000E+00	0,0000E+00	5,0405E-03	4,9662E-03	5,6840E-03	4,2527E-03
$F_{el,TT}$	-6,5052E-05	-4,8245E-06	7,5478E-05	1,4909E-04	-1,3579E-04	-7,6523E-05	-2,9125E-06	5,8271E-05	1,0129E-04	2,0358E-04
$F_{TWW,TT}$	-1,3633E-04	-7,2320E-05	3,0121E-04	3,5376E-04	-2,1371E-04	1,3785E-04	6,9069E-05	1,9392E-05	7,3589E-04	3,9197E-04

MFH	ÜWH	ÜWB	ÜSH	ÜSB	SWX	SSX	WWH	WWB	WSH	WSB
$F_{Heiz,TT}$	1,7589E-03	2,2946E-03	1,8072E-03	1,8481E-03	3,3712E-04	2,7087E-04	5,8914E-03	4,7108E-03	5,7554E-03	3,8752E-03
$F_{el,TT}$	1,2016E-05	2,0910E-06	-1,4781E-05	-1,4781E-05	-3,9592E-05	-4,1577E-05	5,0721E-05	1,7970E-05	3,0872E-05	-8,8641E-07
$F_{TWW,TT}$	1,1700E-05	2,8462E-06	2,8462E-06	1,5635E-05	-5,4213E-05	-4,5359E-05	2,7441E-05	2,6457E-05	1,9570E-05	1,3668E-05

Mittlere Tagestemperatur	11,7	9,9	10,9	10,5	16,4	17	-0,3	3	0,6	3,9
Anzahl Typtage	33	87	8	19	71	10	28	83	5	21

Abb. 5.29 Faktoren der Klimazone TRY03 für die Stadt Oldenburg zur Berechnung des jährlichen Bedarfs an thermischer Energie ($F_{Heiz,TT}$), elektrischer Energie $F_{el,TT}$ und Warmwasser $F_{TWW,TT}$ (wird nicht benötigt) für Einfamilienhäuser (EFH) und Mehrfamilienhäuser (MFH) für die jeweiligen Typtage.

Der Tagesenergiebedarf der Typtage wird als Anteil vom Jahresenergiebedarf für die thermische Energie wie folgt berechnet:

$$Q_{Heiz,TT} = Q_{Heiz,a} \cdot F_{Heiz,TT} \tag{5.17}$$

Dabei ist $Q_{Heiz,TT}$ der thermische Tagesenergiebedarf für den Typtag, $Q_{Heiz,a}$ der jährliche Bedarf an thermischer Energie für das simulierte Gebäude und $F_{Heiz,TT}$ der gegebene Faktor (siehe Abbildung 5.29) für den Typtag. Der Tagesbedarf an elektrischer Energie wird bestimmt gemäß:

$$W_{TT} = W_a \cdot (\frac{1}{365} + N_{WE} \cdot F_{el,TT}) \tag{5.18}$$

Hier ist W_{TT} der berechnete Wert für den Typtag, W_a der jährliche Bedarf an elektrischer Energie, N_{WE} die Anzahl der Wohneinheiten, die für das System auf einen fixen Wert festgelegt wird und $F_{el,TT}$ der gegebene Faktor (siehe Abbildung 5.29) für den Typtag.
Mit dem berechneten täglichen Bedarf an elektrischer und thermischer Energie können mit Hilfe der Referenzlastprofile, die von der VDI-Norm für die zwei Gebäudetypen (Einfamilien- und Mehrfamilienhaus) und für die jeweiligen Typtage zur Verfügung gestellt werden, die Tagesbedarfsgänge ermittelt werden. Ein beispielhaftes Referenzlastprofil ist in Abbildung 5.30 gegeben.

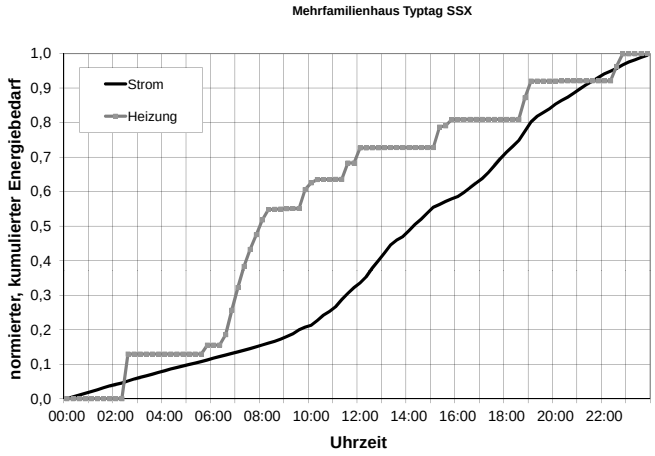


Abb. 5.30 Referenzlastprofil eines Mehrfamilienhauses für den Typtag SSX. Die Werte sind normiert und kumuliert für elektrische (Strom) und thermische Energie (Heizung).

Die Werte sind in einem Takt von 15 Minuten für den entsprechenden Typtag in Tabellenform angegeben. Jedes Element wird mit dem Tagesenergiebedarf der be-

treffenden Typtagkategorie und der zugehörigen Energieform multipliziert. Daraus folgt für jeden Zeitabschnitt der absolute Energiebedarf in kWh für das simulierte Gebäude. Diese Berechnung lässt sich auch für mehrtägige Simulationen durchführen.

Kapitel 6
Realisierung

In diesem Kapitel wird die Umsetzung unseres Entwurfs dargelegt. Dazu werden wichtige Implementierungsdetails begründet erläutert und Kernelemente der Umsetzung herausgestellt. Die Struktur dieses Kapitels orientiert sich dabei am Entwurf (Abschnitt 5), wobei hier die Bottom-Up-Idee durch Zeitgeber, Datenbank und Logging unterbrochen wird. Dies ist der Fall, da diese Systemteile für die weitere Umsetzung eine zentrale Rolle spielen. Weiterhin ist die Reihenfolge einiger Kapitel durch Abhängigkeiten getauscht, um einen besseren Lesefluss zu ermöglichen.

Die Realisierung beginnt in Abschnitt 6.1 mit der Beschreibung, wie die entworfene Architektur in Code abbildbar ist, um einen Rahmen für die Systemimplementierung zu bieten. Fortgefahren wird in Abschnitt 6.2 mit der Realisierung der Komponentenkommunikation. Anschließend betrachtet Abschnitt 6.3 die Umsetzung der Konzepte der Systemkommunikation. Hierauf aufbauend folgt mit Abschnitt 6.4 der Zeitgeber, welcher den wichtigen Schritt der Synchronisierung von EMS und SF erklärt. Danach ist mit Abschnitt 6.5 die Implementierung des Loggers gegeben um ausführliche Logs zu ermöglichen. Darauf folgend wird in Abschnitt 6.6 die Datenbank erläutert, da das Speichern von Run- und Szenariodaten für alle folgenden Systeme essentiell ist. Nun kann das Sampling in Abschnitt 6.7 folgen, welches von den Komponentenmodellen aus Abschnitt 6.8 begleitet wird. Diese beiden Kapitel beschreiben zunächst die Umsetzung der Grundlagen und der Samplingmodelle, worauf die Implementierung der eigentlichen Komponentenmodelle folgt. Hierauf aufbauend folgt mit Abschnitt 6.9 die Implementierung der Optimierungskomponente, welche das Kombinieren und Bewerten der Samplingdaten beinhaltet. Im Anschluss daran wird in Abschnitt 6.10 die Umsetzung des Simulationsframeworks beschrieben, durch welches die EVS-Komponenten während der Simulation mit korrekten Simulationsdaten, zum Beispiel Wetterdaten, versorgt werden. Zuletzt wird die Implementierung der Benutzeroberfläche mit Abschnitt 6.11 beleuchtet.

6.1 Abbildung Architektur auf C++

Um die oben beschriebene Softwarearchitektur möglichst strukturerhaltend auf das zu realisierende Gesamtsystem abzubilden, wurden die von Microsoft Visual Studio bereitgestellten Methoden genutzt. So bietet Visual Studio die Möglichkeit Projekte in einer Solution zusammenzufassen. Eine Solution ist eine Projektmappe, die Projekte in Ordnerstrukturen gliedern kann. [40] Dadurch war es möglich, die Systeme Enegiemanagementsystem, Simulationsframework und Steuerungsebene in drei getrennte Solutions zu gliedern, die jeweils für sich ein Programm repräsentieren. So gibt es beispielsweise eine Solution für das Energiemanagementsystem, die nach der Kompilierung eine ausführbare Datei und benötigte dlls für das System erstellt. Dadurch, dass wir drei Solutions entwickeln, ist die in den Anforderung gegebene Trennung der Systeme möglich.

Ein System spaltet sich in Subsysteme auf. Ein Subsystem ist nur eine logische Trennung, um Module in zusammengehörige Aufgabengebiete zu gliedern. Dies ist in Microsoft Visual Studio durch Ordnerstrukturen in den Projektmappen möglich. Subsysteme sind also kein kompliziertes Gebilde, sondern durch sehr einfach zu erzeugende Ordnerstrukturen abbildbar. Analog zu Subsystemen lassen sich auch Module als Ordner abbilden. Module sind in unserer Lösung Ordner, die keine weiteren Ordner beinhalten.

In einem Modul befinden sich die eigentlichen C++-Projekte wieder, die den Quellcode beinhalten. Ein Modul besteht jedoch nicht nur aus einem Projekt, sondern aus mehreren Projekten. Die Mehrheit der Module besteht dabei aus einem API-Projekt, einem Commons-Projekt und mindestens einem Implementierungsprojekt. API steht für „application programming interface“ und stellt die Schnittstelle des Systems dar, in welcher die Methoden beschrieben werden, die vom Modul nach außen sichtbar anderen Modulen zur Verfügung gestellt werden. Werden in der API selbst geschriebene Datentypen benötigt, werden diese in dem Commons-Projekt des Moduls definiert. Das Commons-Projekt beinhaltet somit alle Datentypen, die auch von anderen Modulen verwendet werden können. In den Implementierungsprojekten befinden sich die Realisierung der in der API definierten Methoden. Die Trennung in API und Implementierung sorgt für eine saubere Trennung von Definition und Realisierung der wichtigen Methoden und ermöglicht es somit erstens eine strukturierte Implementierung und zweitens eine einfache Austauschbarkeit.

Bei den Projekten wiederum gibt es zwei Typen. Die meisten Projekte sind Bibliotheksobjekte, welche dem Programm Methoden und Klassen in DLL-Dateien zur Verfügung stellt. Neben diesen Bibliotheksprojekten gibt es in den drei Systemen jeweils ein Projekt mit einem Einsprungspunkt, der den Start des Systems definiert. Solch ein Projekt erzeugt bei Kompilierung eine ausführbare EXE. Somit lässt sich, wie gefordert, das Energiemanagementsystem unabhängig vom Simulationsframework starten.

6.2 Komponentenkommunikation

Für die Realisierung der Komponentenkommunikation wurde zunächst die Funktionalitäten, Daten an die EVS-Komponenten zu senden beziehungsweise von den Komponenten auszulesen, gebündelt implementiert. Dabei wird für jede EVS-Komponente eine eigenständige Verbindung zu dem jeweiligen OPC-UA-Server aufgebaut, auch wenn sie sich den Server mit anderen Komponenten teilen. Neben dem Schreiben und Lesen von Daten wurde auch eine Funktion integriert, mit der auf Boolean-Variablen der Komponenten gewartet werden kann bis diese den erwarteten Wert annehmen. Mit diesen Funktionen konnte im *EVS-Component-Manager*, der Aufrufe der Komponenten immer für beliebig viele Komponenten gleichzeitig annimmt, realisiert werden.

Der Vorteil, den die Realisierung des *EVS-Component-Mangers* bringen soll, ist die Möglichkeit, die Aufrufe an die EVS-Komponenten parallel auszuführen. Um die parallele Ausführung umzusetzen wurden *ThreadGroups* und *Threads* aus der *Boost*-Bibliothek verwendet. Durch die Verwendung der *ThreadGroups* kann der *EVS-Component-Manger* anfragen mit einer Liste von Komponenten und den jeweiligen Parametern annehmen, die ausgelesen oder beschrieben werden sollen. Dabei wird im *EVS-Component-Manager* wieder ein eigenständiger *Thread* für jede einzelne Komponente erstellt und der *ThreadGroup* hinzugefügt. Sobald alle *Threads* erstellt wurden, muss der *EVS-Component-Manager* lediglich warten bis die *ThreadGroup* ihre Ausführung beendet hat.

Die Ausführung innerhalb der *Threads* für die Komponenten ist dabei meist ein simpler Aufruf einer Funktion des Objekts für die einzelne Komponente. Die einzige Ausnahme bildet die Sampling-Funktion, die durch Anforderung EMS-100A definiert wird, bei der sich die Ausführung in zwei Phasen aufteilt. Die erste Phase ist das Setzen der Sampling-Flag auf *true* und anschließendes Warten bis die Flag wieder auf *false* gesetzt wird. Durch das Setzen dieser Flag wird, wie in Abschnitt 5.7 beschrieben, das Sampling der Komponente angeworfen. Die Komponente setzt nach Abschluss des Vorgangs die Flag wieder auf *false* um dem System mitzuteilen, dass die Samples ausgelesen werden können. In der zweiten Phase muss dann nur noch die Liste der Samples ausgelesen werden. Die Rückgabe der entsprechenden Werte erfolgt dabei direkt in den Objekten, deren *Pointer* den Threads mit übergeben wurden.

6.3 Systemkommunikation

Um die in Abschnitt 5.9 dargestellte Architektur zu realisieren, gibt es diverse Möglichkeiten.

Eine Möglichkeit, diese Kommunikation zu realisieren, ist die direkte Verwendung von *Sockets* ohne Nutzung eines Frameworks. Ein wesentlicher Vorteil ist, dass sich nahezu jede Form von uni- oder bidirektionaler Kommunikation mithilfe von *Sockets* realisieren lässt. Ein entscheidender Nachteil ist jedoch, dass *Sockets* in

verschiedenen Betriebssystemen unterschiedlich realisiert werden - z.B. wird unter Windows die *Windows Sockets API (Winsock)* verwendet, während unixartige Betriebssysteme i.d.R. *BSD-Sockets* verwenden [34]. Somit hätte die Verwendung von *Sockets* eine Plattformabhängigkeit zur Folge. Eine Portierung von Windows auf ein anderes Betriebssystem wäre somit zeitaufwändig.

Eine alternative Variante besteht in der Nutzung eines Frameworks, welches extern plattformunabhängig ist, intern aber Sockets verwendet und somit ebenfalls viele Kommunikationsmöglichkeiten bietet.

Zunächst wurden daher nach einer groben Vorauswahl die folgenden Frameworks betrachtet:

- **Unicomm++** (http://www.libunicomm.org): Unicomm ist ein plattformunabhängiges Framework, welches z.B. vorab definierte XML-Formate zum Datenaustausch bietet. Es unterstützt Microsoft Visual Studio 2010 und ließe sich daher auch mit unserem Projekt verwenden. Während die Kommunikation über das TCP-Protokoll realisiert wird, soll sich der Entwickler auf die Geschäftslogik konzentrieren können und von der Kommunikation abstrahieren können. Neben diesen Vorteilen ist es jedoch kaum dokumentiert. [58]
- **ZeroMQ** (http://www.zeromq.org): ZeroMQ ist ein weiteres Framework, welches den Versand von Nachrichten mittels verschiedenster Netzwerktechnologien (IPC, TCP, TPIC, multicast und inproc) abbildet. Es bietet folgende Vorteile: Es ist plattform- und nahezu programmiersprachenunabhängig, schnell, gut dokumentiert und kann mittels verschiedener Pattern (z.B. Router/Dealer, Publisher/Subscribe) einfach erweiterbare Architekturen abbilden (z.B. die in Abbildung 5.9.1 beschriebene Architektur mit einem zentralen Broker). [28]
- **RabbitMQ** (http://www.rabbitmq.com): RabbitMQ ist ebenfalls ein nachrichtenbasiertes Framework, welches verschiedene Pattern (z.B. Routing, Publisher/Subscribe, RPC) unterstützt. Weiterhin wird JSON unterstützt. Nachteilig ist, dass keine offizielle Version und entsprechend keine Dokumentation für C++ existiert. Es gibt lediglich eine von Nutzern portierte Version. Dies führt zu zusätzlichem Zeitaufwand bei der Einarbeitung bzw. beim Verständnis der Dokumentation. [46]

Ein Vergleich der verschiedenen Frameworks führte letztlich zur Entscheidung für ZeroMQ. Dafür sprachen neben den dargestellten Vorteilen ein großer Anwenderkreis und eine zu *Sockets* ähnliche Verwendbarkeit. Weiterhin wird ZeroMQ unter der Open-Source-Lizenz *LGPL* veröffentlicht [28]. Diese Lizenz ermöglicht die Verwendung von ZeroMQ in proprietären Anwendungen, sodass die im Projekt entwickelte Software (sofern alle verwendeten Frameworks ebenfalls die *LGPL* oder vergleichbare Lizenzen nutzen) auch verkauft werden könnte [14]. Gegen die anderen Frameworks sprach vor allem die unzureichende Dokumentation.

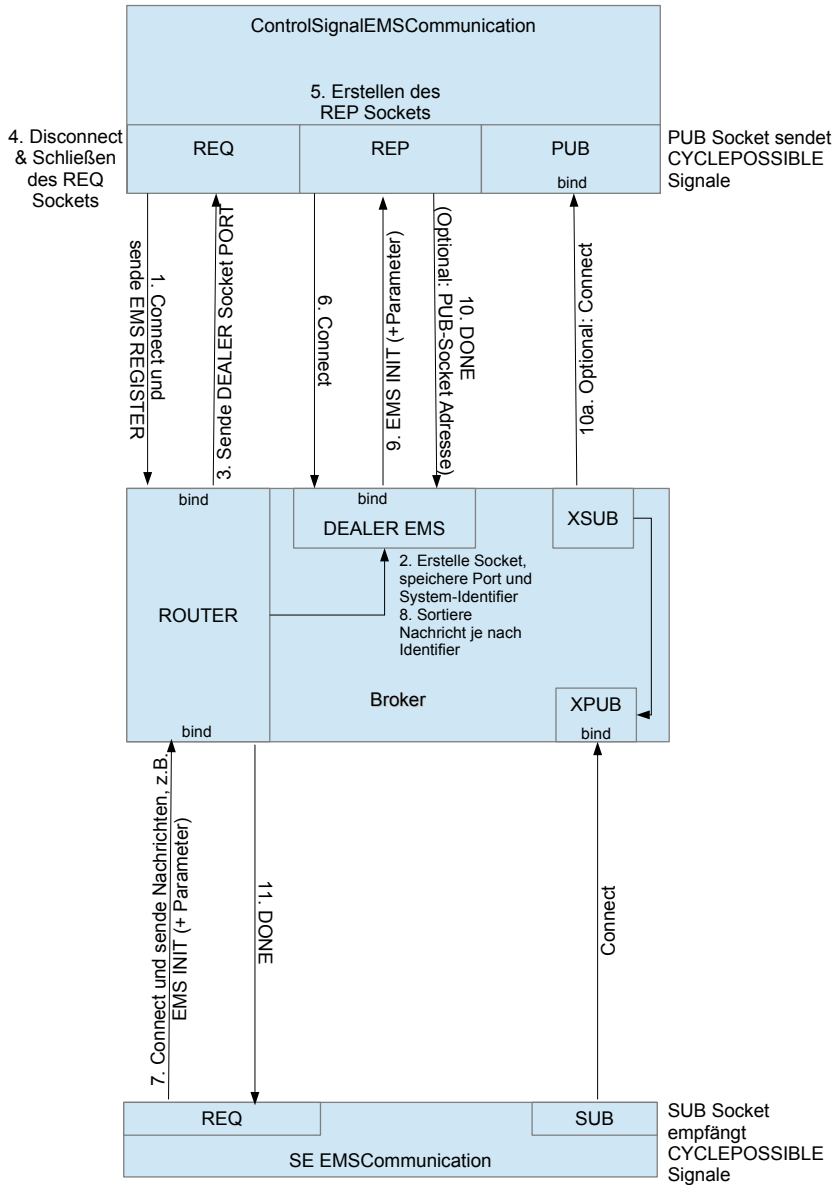


Abb. 6.1 Realisierung der Kommunikation von Steuersignalen (Eigene Darstellung)

6.3.1 Realisierung der Kommunikation von Steuersignalen

Ziel der Realisierung war die Umsetzung der in Abbildung 5.9.1 beschriebenen Architektur, die durch einen Broker (der wie eine Art Postamt funktioniert) gekennzeichnet ist (zur Grundidee eines einfachen Brokers siehe auch [25]).

Die Realisierung dieser Architektur ist am Beispiel des EMS in Abbildung 6.1 dargestellt.

Mittig dargestellt ist die Broker-Klasse, die mittels *Router*- und *Dealer*-Sockets eine hohe Erweiterbarkeit für zusätzliche, kommunizierende Systeme sicherstellt.

Im oberen Bereich dargestellt ist die Klasse *ControlSignalEMSCommunication* (basiert auf dem Interface *ControlSignalSystemCommunication*) des EMS dargestellt. Diese benötigt drei Sockets, ein *Request*-Socket für den Verbindungsaufbau zum Broker, ein *Response*-Socket zur Beantwortung eingehender Steuersignale sowie ein *Publisher*-Socket, um die Steuerungsebene über durchgeführte Zyklen zu informieren.

Im unteren Bereich der Abbildung befindet sich die Klasse *EMSCommunication* der Steuerungsebene, welche auf dem Interface *iCLCommunication* basiert. Diese benötigt ein Request-Socket, um Steuersignale zu senden und Antworten zu empfangen. Außerdem wird ein *Subscribe*-Socket benötigt, um Nachrichten über durchgeführte Zyklen zu empfangen.

Die Funktionsweise dieser drei Klassen wird im Folgenden anhand eines kleinen Beispiels dargestellt. Die Signale werden dabei i.d.R. in Form von Enums realisiert.

1. Zunächst kommuniziert das System (in Abbildung 6.1 das EMS) mithilfe eines *Request*-Sockets mit dem *Router*-Socket des Brokers und übermittelt diesem ein Enum über die Identität des Systems sowie ein REGISTER-Signal.
2. Der Broker überprüft nun, ob bereits ein Socket für das spezifische System besteht. Ist dies nicht der Fall, wird ein *Dealer*-Socket für dieses System erzeugt.
3. Der Broker schickt im Anschluss über das *Router*-Socket den Port des neu eröffneten Sockets.
4. Das *Request*-Socket des Systems disconnected sich und wird geschlossen.
5. Im Anschluss wird im System ein neues *Response*-Socket eröffnet,
6. welches sich mit dem soeben eröffneten *Dealer*-Socket verbindet (mithilfe des mitgeteilten Ports).
7. Die Kommunikation der Steuerungsebene (im Beispiel die Klasse *SIMCommunication*) verfügt über ein *Request*-Socket und verbindet sich mit dem *Router*-Socket des Brokers. Anschließend wird z.B. ein INIT-Signal mit entsprechenden Parametern (ID des Scenarios und des Run-Objektes) in Verbindung mit einem Enum des Systems, welches die Nachricht erreichen soll, gesendet. Kommuniziert ein *Request*-Socket mit einem *Router*-Socket, so wird die Identität des *Request*-Sockets in ZeroMQ automatisch mitgesendet (vgl. hierzu [25]).
8. Im Broker wird nun das Identifizierungs-Enum wie eine Adresse in einem Postamt ausgewertet. Im Beispiel ist die Nachricht an das EMS gerichtet und wird an das *Dealer*-Socket des EMS weitergeleitet.

9. Das *Response*-Socket des EMS empfängt die Nachricht und wertet sie aus. Im Beispiel wird z.B. der Controller des Systems über die Szenario-ID und die zugehörige Run-ID informiert und kann diese für Datenbankanfragen nutzen.
10. Anschließend wird die Nachricht (im Beispiel mit DONE) beantwortet. Optional kann das EMS nun die Adresse eines *Publisher*-Sockets mitsenden. Diese dienen dazu, nach vollständigem Durchlauf einer Optimierung der SE mitzuteilen, dass Daten des jeweiligen Zyklus (dieser wird z.B. durch einen *Cycle* mit entsprechender *Cycle-ID* dargestellt) abgerufen werden können. Wird diese Adresse mit gesendet, verbindet sich das *XSUB*-Socket des Brokers mit dem *PUB*-Socket des Systems. Eingehende Nachrichten werden an das *XPUB*-Socket weitergeleitet, gesendet und anschließend von einem *SUB*-Socket der Steuerungsebene empfangen.
11. Die Antwort (DONE) geht zunächst im *Dealer*-Socket ein. Im Broker wird sie über das *Router*-Socket weiterversendet. Durch beim *Request* automatisch mitgesendete Identität (vgl. Schritt 7) kann das Router-Socket die Antwort an den korrekten Empfänger, d.h. an das ursprüngliche *Request*-Socket, ausliefern.

Anschließend können weitere, notwendige Steuersignale, etwa zum Start der Optimierung, folgen. Diese werden ebenfalls mit einer Antwort bestätigt.

Um die Abbildung 6.1 übersichtlicher zu gestalten, wurden Status-Abfragen abfragen nicht mit eingezeichnet: Vor jedem Request wird in der Steuerungsebene mittels eines HANDSHAKE-Signals zunächst geprüft, ob das angesprochene System antwortet und erst bei erfolgter Antwort weitere Signale gesendet.

Die etwas aufwändige Implementierung des Brokers bietet folgende Vorteile:

- Es können beliebige, weitere Systeme über den Broker kommunizieren (z.B. die Zeitgeber des EMS und des SF, vgl. Abschnitt 6.4), da für jedes hinzukommende System ein separates Socket erzeugt wird und mithilfe der Kombination von *Router*- und *Dealer*-Sockets auch bei vielen Systemen sichergestellt wird, dass Anfragen und Antworten die korrekten Empfänger erreichen.
- Theoretisch wäre auch möglich, dass sich zwei Systeme mit dem gleichen Identifizierungs-Enum beim Broker anmelden. Diesen würde dann das gleiche *Dealer*-Socket zugeteilt und beide Systeme würden die Nachrichten gleichzeitig erhalten.
- Da das SF und das EMS unterschiedliche Identifizierungs-Enums verwenden, können Steuersignale an beide Systeme getrennt gesendet werden.
- Die Systeme können die Steuerungsebene unidirektional über ausgeführte Zyklen informieren, ohne auf eine Antwort warten zu müssen.
- Für die Kommunikation mit einem System wird lediglich die Adresse des Brokers sowie das Identifizierungs-Enum des Systems benötigt.

Nachteilig ist die bereits im Entwurf dargestellte Abhängigkeit der Kommunikation von der Funktion des Brokers. Die Kommunikation wurde zudem unter der Annahme implementiert, dass sich die im Netzwerk befindlichen Rechner in einer Sicherheitszone befinden und die verwendeten Ports von außerhalb nicht zugänglich sind (siehe auch [6]). Für die Verwendung in offenen Netzwerken sollten dementsprechend Sicherheitsmechanismen integriert werden: Ansonsten könnten Angreifer sich

mittels REGISTER-Nachrichten Zugang zu Sockets verschaffen oder sich weitere Sockets erzeugen lassen.

Im Anhang befinden sich zwei Sequenzdiagramme, die den Ablauf der Kommunikation verdeutlichen. Das erste stellt den Verbindungsaufbau der Systeme sowie ein INIT-Signal dar (vgl. Abbildung A.1), während das zweite beispielhaft das Senden und Empfangen eines START-Signals beschreibt (vgl. Abbildung A.2).

6.3.2 Realisierung der Kommunikation von Logging-Nachrichten

Die Umsetzung der in Abschnitt 5.9.2 beschriebenen Architektur ist in Abbildung 6.2 dargestellt. Der wesentliche Bestandteil ist wie im Entwurf beschrieben der Collector, der über vier Sockets verfügt, deren Funktion jeweils im Folgenden kurz erläutert wird (vgl. hierzu auch [25]).

Das *Response*-Socket oben (REP) dient dem Verbindungsaufbau der Systeme, die Log-Nachrichten an den Collector senden möchten (z.B. EMS oder SF). Auf diesem Socket soll dementsprechend ein *Request* des Systems, welches Log-Nachrichten senden möchte, eingehen (Schritt 1). In diesem Request teilt das System dem Collector die Adresse eines *Publisher*-Sockets (PUB) mit: Dieses PUB-Socket sendet unidirektional Log-Nachrichten, die wiederum von einem *Subscriber*-Socket (SUB) empfangen werden können. In diesem Fall wird dazu im Collector ein *XSUB*-Socket benötigt, welches *Subscription-Forwarding* betreiben kann: Jedes Log besteht aus zwei Nachrichten, die erste beinhaltet ein Subscription-Tag (der Systemname, z.B. SE oder EMS), während die zweite Nachricht aus dem eigentlichen Inhalt besteht. Das Subscription-Forwarding des *XSUB*-Sockets ermöglicht nun, dass später nach diesen Tags gefiltert werden kann. Das XSUB-Socket verbindet sich also nach Eingang der Adresse eines PUB-Sockets mit diesem und empfängt dessen Nachrichten (Schritt 2). Dies wird mit OK bestätigt (Schritt 3).

Wenn das System beendet wird, kann es sich beim Collector abmelden: Dazu wird über das *Request*-Socket ein CLOSE-Signal und erneut die Adresse des *Publisher*-Sockets gesendet (Schritt 4). Das *XSUB*-Socket des Collectors beendet die Verbindung zum *Publisher*-Socket des Systems (Schritt 5) und bestätigt dies anschließend mit OK (Schritt 6). Diese Abmeldung dient somit dazu, dass das Schließen der Verbindung durch das *XSUB*-Socket sichergestellt wird.

Im unteren Teil des Collectors befinden sich wiederum zwei Sockets: Ein *Response*-Socket (REP) für den Verbindungsaufbau und ein *XPUB*-Socket, welches die Logs (und die Subscription-Tags) an ein bzw. mehrere *Subscriber*-Sockets weiterleitet. Somit kann sich sowohl der Controller bzw. die GUI der SE mit dem Collector verbinden, als auch z.B. eine weitere Klasse, die sämtliche Logs in Dateien schreibt. Dies ist jedoch aufgrund des Loggingkonzeptes, welches sämtliche Logs bereits in Dateien schreibt, nicht notwendig. Der vorherige Verbindungsaufbau mithilfe des *Response*-Sockets ist notwendig, damit der Collector weiß, wann eine Verbindung hergestellt wurde und die Nachrichten so lange speichern kann. Dies stellt sicher, dass auch alle Nachrichten durch die *Subscriber* empfangen werden. Die

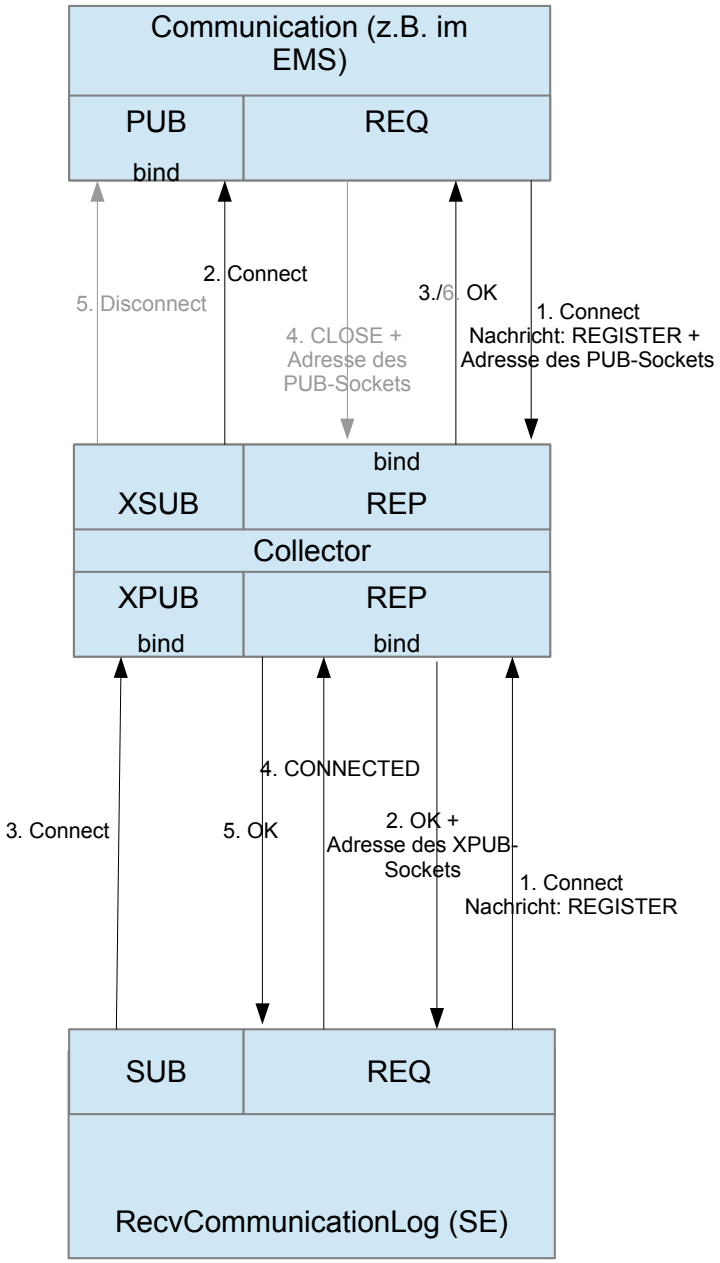


Abb. 6.2 Realisierung der Kommunikation von Log-Nachrichten (Eigene Darstellung)

Anmeldung und Abmeldung der Steuerungsebene funktioniert dabei ähnlich wie in den jeweiligen Systemen. Die Steuerungsebene meldet sich per REGISTER-Signal

an (Schritt 1), erhält anschließend eine Bestätigung sowie die Adresse des *XPUB*-Sockets des Collectors (Schritt 2). Daraufhin verbindet sich das *Subscriber*-Socket mit dem *XPUB*-Socket (Schritt 3) und bestätigt dies mittels des Signals CONNECTED (Schritt 4). Da auf jedes *Request* eine *Response* folgen muss, sendet der Collector erneut eine Bestätigung (Schritt 5). Wenn sich die Steuerungsebene abmeldet (nicht in der Abbildung dargestellt), sendet sie ein CLOSE-Signal, welches mittels OK bestätigt wird. Die Abmeldung ist nicht zwingend notwendig, da sie lediglich dazu dient, dass die Logs im Collector wieder zwischengespeichert werden.

Die Systeme, die Log-Nachrichten senden möchten (z.B. EMS oder SF), benötigen somit zwei Sockets, ein *Request*-Socket für den Verbindungsaufbau und ein *Publisher*-Socket zum unidirektionalen Versand der Logs.

Die Systeme, die Log-Nachrichten empfangen möchten (z.B. die die Klasse *RecvCommunicationLog* der Steuerungsebene), benötigen ebenfalls zwei Sockets: Ein *Request*-Socket für den Verbindungsaufbau sowie ein *Subscriber*-Socket, um die vom Collector gesammelten Logs zu empfangen.

6.4 Zeitgeber

Nach dem in Abschnitt 5.12 beschriebenen Entwurf wurde der Zeitgeber mithilfe des dort beschriebenen, erweiterten Observer-Patterns umgesetzt.

Diese Lösung hat zur Folge, dass die Aufrufe der einzelnen Komponenten die Ausführung eines Zyklus blockieren. Damit der Zeitgeber auch auf die Zeit achten kann, muss deshalb in einem nebenläufigen Prozess ein Timer ausgeführt werden. Auf diesen Timer muss im Prozess des Zeitgebers gewartet werden, damit ein Takt nicht zu früh beendet wird. Neben der Möglichkeit auf den Timer zu warten, muss der Timer, sobald er durchgelaufen ist, in der Lage sein, die Ausführung des Taktes korrekt zu beenden. Dies verhindert eine Überschreitung der Taktzeit. In der „cycle()“-Methode, die der „update()“-Methode des Observer-Patterns entspricht, wird - neben einer ID für den Takt zur eindeutigen Identifizierung - die Ausführungszeit übergeben und beachtet.

Weiterhin muss noch die Verbindung zu der Proxy-Clock im Simulationsframework nach dem Proxy-Pattern realisiert werden.

Synchronisierung und Kommunikation der Zeitgeber

Für die Synchronisierung und die Kommunikation der Zeitgeber wurde der in Abschnitt 5.12 beschriebene Entwurf umgesetzt.

Der Zeitgeber des Simulationsframeworks fungiert als Observer des Zeitgebers im EMS an, sofern das Simulationsframework aktiv ist und dort über die Systemkommunikation ein INIT-Signal eingeht. Somit wird wie im Entwurf dargelegt ein modifiziertes Proxy Pattern verwendet (vgl. [15, S. 207ff.]). Beide Zeitgeber nutzen das „ClockGeneratorInterface“, sodass eine Austauschbarkeit gewährleistet ist.

In Abbildung 6.3 ist zunächst der Verbindungsaufbau der Zeitgeber dargestellt. Die Controller der beiden Systeme implementieren jeweils das „ClockObserver“-Interface. Der Controller des EMS registriert sich direkt beim Taktgeber. Die Kommunikation der Zeitgeber erfolgt im EMS und im SF jeweils über eine Klasse „ClockCommunication“. Diese nutzen jeweils den in Abschnitt 6.3.1 beschriebenen Broker, um eine Verbindung aufzubauen und zu kommunizieren. Die ProxyClock registriert sich beim Taktgeber des EMS, um die ClockCommunication über eingehende Zyklen zu informieren. Der SIMController registriert sich beim ProxyClockGenerator des Simulationsframeworks.

In beiden Systemen wird eine ClockCommunication-Klasse, die für die Kommunikation der Zeitgeber zuständig ist, in einem Thread gestartet. Dies stellt sicher, dass das Warten auf Nachrichten das übrige System nicht blockiert. Geht nun im Simulationsframework ein „INIT“-Signal ein, informiert der Controller die ClockCommunication mittels eines „SCENARIO“-Signals und der Szenario-ID darüber. Dies geschieht über Sockets, da dies der von ZeroMQ empfohlene Weg ist, Daten zwischen Threads auszutauschen (vgl. [25]). Gleiches geschieht im EMS, wo ebenfalls der Controller die ClockCommunication über das Szenario informiert.

Nach Eingang des SCENARIO-Signals in der ClockCommunication des Simulationsframeworks sendet diese ein „REGISTER“-Signal und die Szenario-ID an die ClockCommunication des EMS. Die Szenario-ID wird dort gespeichert und der Eingang des „REGISTER“-Signals bestätigt.

In Abbildung 6.4 wird nun der Ablauf eines Zyklus dargestellt. Zunächst ist dazu ein „START“-Signal der Steuerungsebene erforderlich, welches im EMS eingeht. Im Controller wird daher die Methode „receiveStartSignal()“ aufgerufen, die aus dem spezifischen, mittels RunID identifizierten RunData-Objekt die Cycletime und damit den Zeitraffer ausliest. Anschließend wird die Methode „startClock()“ des Taktgebers aufgerufen und die Cycletime übergeben. Dort wird in einem Thread die Methode „executeCycles()“ aufgerufen, die bis zum Eingang eines STOP-Signals mithilfe der „singleCycle()“-Methode Zyklen ausführt.

Innerhalb der „singleCycle()“-Methode werden der Reihe nach die Observer informiert und dort jeweils die „cycle()“-Methode aufgerufen. Als erstes wird die ProxyClock informiert.

Diese benachrichtigt wiederum über ein *Request* (ein „CYCLE“-Signal und die zugehörige ID) die ClockCommunication über den Zyklus. Die ClockCommunication des EMS sendet daraufhin ein gleich aufgebautes *Request* an die ClockCommunication des Simulationsframeworks weiter. Dort wird bei Eingang der Nachricht der ProxyClockGenerator informiert und ebenfalls die „singleCycle()“-Methode aufgerufen. Der ProxyClockGenerator informiert wiederum seine Observer. Demnach wird die „cycle()“-Methode des SIMControllers aufgerufen. Nach dem Durchlaufen beider Methoden und somit des Zyklus im Simulationsframework beantwortet die SF-ClockCommunication das *Request* und gibt „OK“ zurück. Die EMS-ClockCommunication gibt daraufhin der ProxyClock „CYCLEOK“ zurück. Wäre das Simulationsframework nicht aktiv und demnach nicht bei der ClockCommunication registriert, würde diese sofort „CYCLEOK“ zurückgeben und damit ein Blockieren verhindern.

Im Anschluss werden im EMS vom ClockGenerator die weiteren Observer, etwa der EMS-Controller, informiert und dort die „cycle()“-Methode aufgerufen.

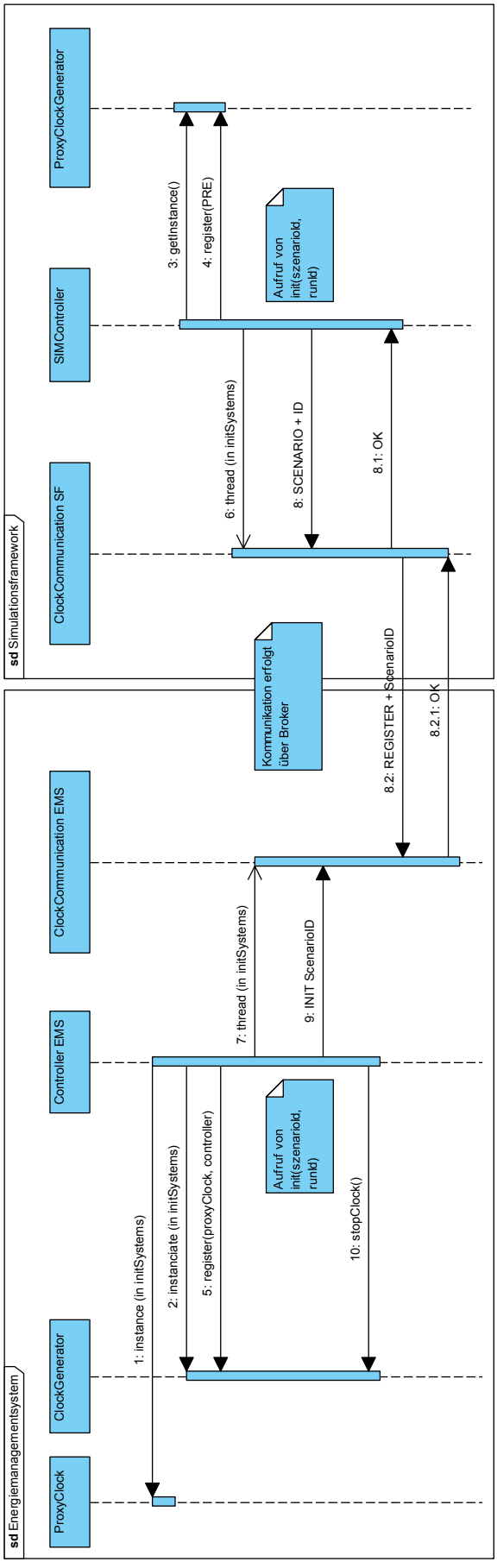


Abb. 6.3 Sequenzdigramm des Verbindungsaufbaus der Zeitgeber der Systeme

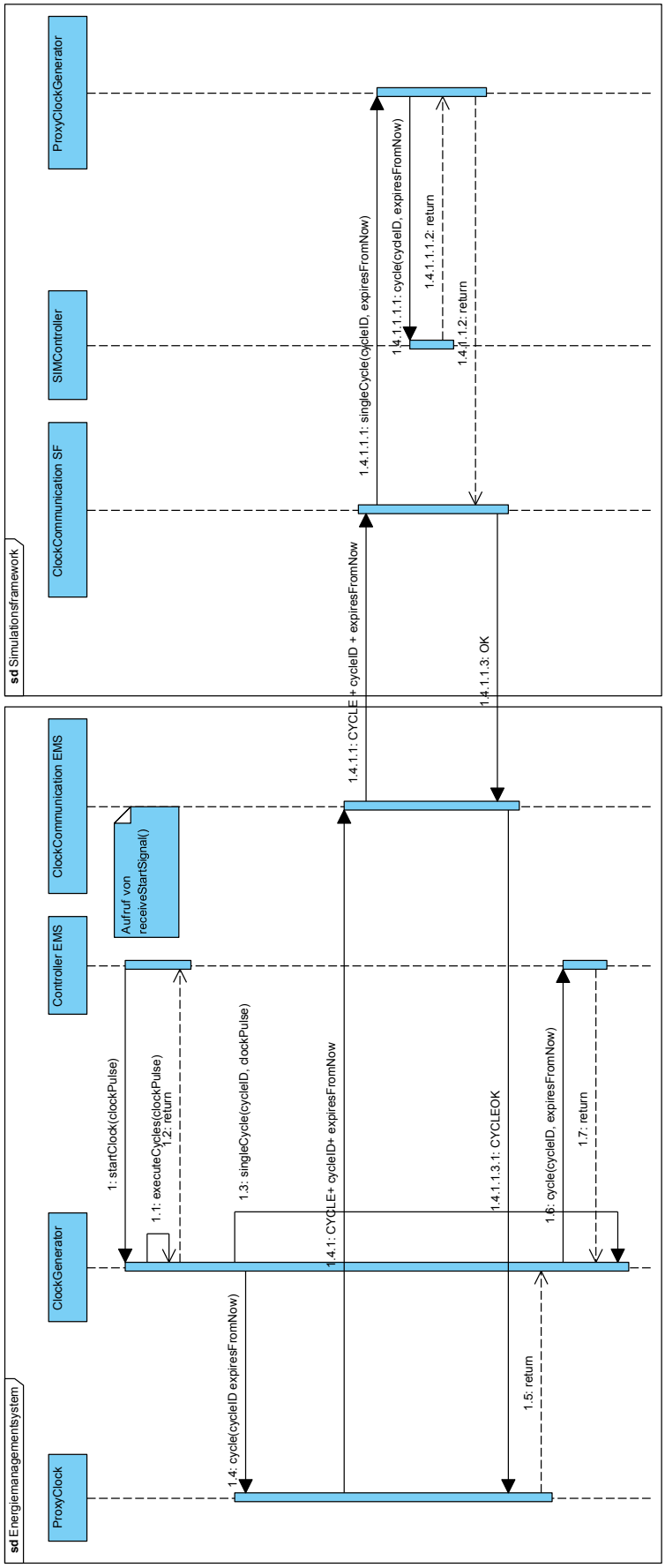


Abb. 6.4 Sequenzdigramm der Zeitgeber während eines Zyklus

6.5 Logging

Im Kapitel Abschnitt 5.14 wurde das Konzept für das Logging vorgestellt. Dabei wurde von der eigentlichen Implementierung eines Loggers abstrahiert. Die Realisierung wird im Folgenden begründet erläutert.

Ein Logger soll die Möglichkeit bieten, sowohl Benutzer über eine GUI zu informieren als auch im Falle der Nichtverfügbarkeit einer grafischen Schnittstelle z.B. beim Absturz des Systems Informationen in Dateien festzuhalten. Ersteres ist durch unser Observerkonzept möglich. Dazu wird jeweils ein Observer beim LoggerObservable registriert, welcher Nachrichten an die Steuerungsebene weiterleitet. Im Detail registriert sich wie in Abschnitt 5.14.1 und Abschnitt 5.9.2 beschrieben in jedem System ein LoggerObserver, der die Logs an den Collector übermittelt. Die Implementierung des LoggerObservers erfolgt in den jeweiligen Kommunikationsklassen des Systems. Im Collector werden die Nachrichten gesammelt und bei Vorhandensein einer Steurungsebene übertragen. Letzteres (das Festhalten der Informationen in Dateien) muss vom Logger selbst ermöglicht werden, da Nachrichten möglichst schnell in Logdateien gespeichert werden sollten. Wenn das System abstürzt, müssen Nachrichten vor dem Absturz als Debuggrundlage abrufbar sein. Da der Logger Nachrichten als erstes erhält, sollte der Logger diese also sofort abspeichern.

Da ausreichend Fremdbibliotheken für das Logging existieren, die auch das Abspeichern von Daten in Logdateien ermöglichen, war es nicht nötig den Logger selbst zu schreiben. Viele dieser Fremdbibliotheken haben den Vorteil, dass sie erprobt sind, bereits Konfigurationsmöglichkeiten aufweisen und thread-safe sind. Dies sind Aspekte, die zu einem großen Aufwand bei einer eigenen Implementierung führen. Eine Fremdbibliothek, welche diese Funktionen anbietet, ist die von uns verwendete Bibliothek easylogging++. [32] Diese hat im Vergleich zu anderen betrachteten Loggern den Vorteil, dass sie eine flexible MIT-Lizenz verwendet, die auch das Anpassen des Loggers erlaubt. Dies haben wir genutzt, um die Benachrichtigung des LoggerObservables bei Erhalt einer Nachricht zu ermöglichen.

Easylogging++ ermöglicht es Lognachrichten unterschiedlicher Kategorien wie Fehler, Information, Warnung usw. zuzuordnen und diese in einer Textdatei mit Berücksichtigung einer zuvor festgelegten Form und Filterung zu speichern. Der Benutzer kann also flexibel einstellen, welche Informationen für ihn entscheidend sind. Dabei kann im Programmcode sehr schnell eine Lognachricht erzeugt werden. Beispielsweise kann folgendermaßen eine SQLException geloggt werden:

```
1 LOGERROR << "SQLException: " << e.what()
2         << " (MySQL error code: " << e.getErrorCode()
3         << ", SQLState: " << e.getSQLState() << " )";
```

Einzelne Nachrichtenteile auch unterschiedlicher Datentypen können dabei mittels << konkateniert werden. Somit lassen sich benötigte Nachrichten ohne großen Aufwand loggen.

6.6 Datenbank

In diesem Kapitel soll kurz auf die Implementierung des Datenbankentwurfs aus Abschnitt 5.13 eingegangen werden. Für alle drei Systeme (EMS, SF und SE) wurde ein gemeinsamer Data-Access-Layer implementiert. Positiv ist daher eine einfache Veränderung der Zugriffsmethoden auf die Datenbank an nur einer zentralen Stelle.

Zur Konfiguration der Datenbank-Verbindung des Systems wurde eine Konfigurationsdatei angelegt. In ihr muss der Nutzer die Adress- und Benutzerinformationen eingeben.

Durch Verwendung einer API zur Kapselung ist der Austausch des Data-Access-Layers ebenfalls möglich. Dadurch können bei späterer Veränderung des Systems auch andere DBMS mit entsprechend entwickeltem Data-Access-Layer verwendet werden.

Die genaue Struktur der Datenbanken war von Beginn an nicht komplett festgelegt, da nicht alle zu speichernden Daten bekannt waren. Daher sind die Datenbanken historisch weiter gewachsen, wobei die Grundstruktur bereits festgelegt war. Der Wachstum hat sich hauptsächlich bei den zu speichernden Laufdaten bemerkbar gemacht, dabei ist uns unsere generische Haltung der Daten zu Gute gekommen.

6.7 Sampling

Im folgenden Kapitel wird unsere Umsetzung des Samplingkonzepts beschrieben. Dazu wurden für die EVS-Komponenten jeweils ein Customer Energy Manager umgesetzt. (Siehe Abschnitt 5.7) Dieser besteht zum einen aus dem Manager der EVS-Komponente, welcher der Verwaltung der EVS-Komponente bzw. Ausführung der simulierten EVS-Komponente dient, und zum anderen aus der Samplingkomponente, welche die Samples generiert.

Die beiden Bestandteile wurden mit einer Sprache für PLC (deutsch: SPS) realisiert, welche von Beckhoff bereitgestellt wird[2]. Diese Sprache setzt den IEC 61131 Standard um und dient somit der Programmierung für eine Speicherprogrammierbare Steuerung, also der Regelung und Steuerung von Geräten. [47] Das CEM ist somit auch auf den Kopplern realisierbar und es besteht die Möglichkeit ohne großen Aufwand auf die Twincat-Module zuzugreifen, d.h. Input Variablen zu beschreiben, Output-Variablen auszulesen und einen Zyklus zu starten. Ein weiterer Vorteil ist, dass PLC leicht um OPC-UA erweiterbar ist. So können Variablen durch einfache Annotationen für den Zugriff über OPC UA freigegeben werden. Somit ist ein einfacher Zugriff auf Parameter, Flags und die Samplearrays des CEM möglich.

Eine Alternative zur direkten Umsetzung in PLC wäre es, den CEM in einem System außerhalb der Koppler umzusetzen. Dadurch wäre die Programmiersprache frei wählbar und die Zwischenschicht hätte nicht nur eine logische Trennung von den EVS-Komponenten, sondern wäre auch örtlich trennbar gewesen. Da die Möglichkeiten über PLC jedoch schon gegeben waren und die Alternative einen unnötig

hohen Arbeitsaufwand bedeutet hätte, haben wir uns dafür entschieden mittels PLC die Steuerung der Twincat-Module zu ermöglichen.

Die Bestandteile - Manager der EVS-Komponente und die Samplingkomponente - wurden in zwei getrennten Funktionsblöcken geschrieben. Diese Funktionsblöcke ähneln sowohl Funktionen als auch Klassen in den objektorientierten Sprachen. Einerseits können sie eine einfache Funktion realisieren. Andererseits ist es aber auch möglich, ihnen andere Funtkionsblöcken und Variablen gewissermaßen als Methoden und Attributen zuzuordnen. Die Steuerung dieser Funktionsblöcke ist über die bereits in Abschnitt 5.7 beschriebenen Flags möglich, welche über OPC-UA vom EMS und SF beschrieben werden können. Die Kopie der EVS-Komponente wurde als TwinCat-Modul realisiert. Dabei ist die Kopie einfach eine weitere Instanziierung des Matlabmoduls einer EVS-Komponente, die keiner weiteren Anpassung benötigt.

In den folgenden Unterabschnitten werden die einzelnen Umsetzungen der Samplingalgorithmen für die EVS-Komponenten beschrieben.

6.7.1 Samplingkomponente Batterie

Die Samplingkomponente der Batterie setzt das in Abschnitt 6.7 erläuterte Samplingkonzept für das in Abschnitt 6.8.2 vorgestellte Modell einer Batterie um. Ergänzend zu dem vorgestellten Konzept ist die Batterie mit einem Variationspunkt für die Evaluation ausgestattet, um überprüfen zu können, welchen Einfluss das Sampling der Batterie auf die Optimierung hat. Über diesen Variationspunkt kann der Modus des Samplings ausgewählt werden. Diese Modi sind ein Default Modus, zwei Modi, welche die relative Grundlast berücksichtigen und ein Modus, der die gegenwärtige Last im Verhältnis zur minimalen, maximalen und durchschnittlichen Last berücksichtigt. Bevor im folgenden genauer auf diese Modi eingegangen wird, wird zunächst Allgemeingültiges zu den Modi erläutert.

Jeder der hier dargestellten Modi berücksichtigt zunächst nicht den aktuellen Zustand der Batterie, sondern nutzt andere Heuristiken. Dies wird möglich durch das Verhalten der Batterie bei Fehlerzuständen, wie in Abschnitt 6.8.2 beschrieben. Sollte versucht werden mehr Spannung aus der Batterie zu entnehmen, als zu dem Zeitpunkt möglich oder mehr Spannung an die Batterie weitergegeben werden, als zu dem Zeitpunkt möglich, so zeigt die Batterie dies über einen Validierungswert an. Wenn eine negative Validierung während des Samplings auftreten sollte, wird der letzte Samplingschritt rückgängig gemacht. Damit hierbei nicht viele Fehlversuche und eine möglicherweise sehr lange Berechnungszeit entstehen, wird in dem neuen Berechnungsschritt für das Intervall die von der Batterie gelieferte tatsächliche Eingangsleistung verwendet. Mehr zu der tatsächlichen Eingangsleistung kann in Abschnitt 6.8.2 gefunden werden. Auf diese Weise entsteht ein robustes Sampling ohne Berücksichtigung des aktuellen Zustandes der Batterie.

Der Default Modus erzeugt ein Sample, bei dem ein zufälliger Wert im Bereich der möglichen Werte, also zwischen maximaler Entladung und maximaler Beladung der Batterie, ausgewählt wird, wie in Abbildung 6.5 verdeutlicht. Dieser Modus ist

als Referenzmodus für die Evaluation vorgesehen. Jedes sinnvolle Sampling der Batterie muss besser als dieses zufällige Sampling sein.

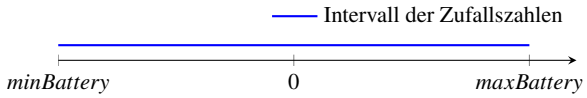


Abb. 6.5 Sampling Modus 0 (Default Sampling Modus)

Die weiteren Modi des Samplings orientieren sich an der Lastprognose für die nächsten bekannten Intervalle, also in der Regel der nächsten 24 Stunden. Der erste und zweite Modi orientieren sich an der maximalen Tageslast und bilden mit der Last des jeweilige Intervall (currentLoad) die relative Last, also Last im Verhältnis zur maximalen Last, wie in Abbildung 6.6 und Abbildung 6.7 zu sehen. Hinter diesen Modi steht die Annahme, dass bei einer im Verhältnis zur maximalen Tageslast hohen Last in einem Intervall die Batterie wahrscheinlich Leistung abgeben sollte, während sie bei einer niedrigen Last im Verhältnis zur maximalen Tageslast vermutlich Leistung speichern sollte. Bei dem ersten dieser beiden Modi wird bei hoher Grundlast, in dem Zeitraum für den das Sample erstellt wird, die Batterie entladen, während bei niedriger Last die Batterie aufgeladen wird. Bei dem zweiten dieser Modi wird genau umgekehrt vorgegangen. Der zweite Modus hat keine praktische Relevanz, soll aber zeigen, dass es sich bei dem ersten Modus um einen sinnvollen Modus handelt. Der zweite Modus sollte zudem schlechter als der zufällige Modus sein. Wenn dies nicht der Fall ist, dann ist die für getroffene Annahme, dass die Last im Verhältnis zur maximalen Tageslast als Anhaltspunkt für das notwendige Verhalten der Batterie dienen kann, widerlegt bzw. die Annahme muss verfeinert werden.

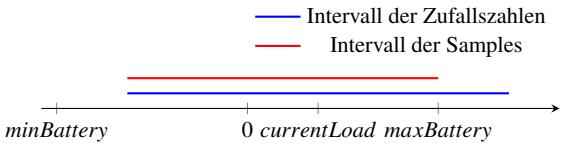


Abb. 6.6 Sampling Modus 1 (Sampling Modus Gegenwärtiger Verbrauch)

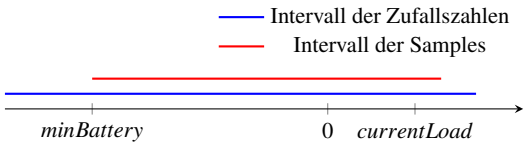


Abb. 6.7 Sampling Modus 2 (Sampling Modus Umgekehrter Gegenwärtiger Verbrauch)

Bei beiden Modi können die Intervalle der Zufallszahlen außerhalb des Ladungsbandes der Batterie liegen, wie in Abbildung 6.6 im Intervall der Zufallszahlen verdeutlicht. Ein Teil des Intervalls liegt oberhalb der maximalen Eingangsleistung der Batterie. Diese Bereiche werden beim Sampling weggeschnitten, indem jeder Wert größer als die maximale Beladung die maximale Beladung bzw. bei Werten kleiner als der minimalen Beladung die minimale Beladung gewählt wird.

Die relative Last wird mit einem zufälligen Faktor manipuliert, um über die Samples eine gewisse Streuung zu erzeugen und der Optimierung damit einen Spielraum zu bieten. An dieser Stelle bietet es sich für tiefer gehende Evaluationen an, einen weiteren Variationspunkt anzubieten. So könnte zum Beispiel ein Faktor ergänzt werden, wie groß der Einfluss der Last ist. In den beiden bisherigen Optimierungen werden lediglich die Werte 1 bzw. -1 angenommen, aber es ist auch jeder Wert dazwischen möglich. Es könnte evaluiert werden, ob hier ein optimaler Faktor existiert oder ob ein optimaler Faktor in einem bestimmten Wertebereich liegt.

Der dritte Modus orientiert sich an den minimalen (minL), maximalen (maxL) und durchschnittlichen (aL) Lastwerten innerhalb der Lastprognose, wie in Abbildung 6.8 illustriert. Hinter diesem Sampling steht die Annahme, dass nicht das Verhältnis zur maximalen Tageslast optimale Aussagen über das Sollverhalten der Batterie prognostiziert, sondern dass ein Verhältnis zur minimalen, durchschnittlichen und maximalen Last entscheidend ist. Diese These stützt sich auf die Annahme, dass die Batterie gerade dann aufgeladen werden muss, wenn die Grundlast niedriger als die durchschnittliche Grundlast ist und Leistung abgeben muss, wenn die Last höher als die durchschnittliche Tageslast liegt.

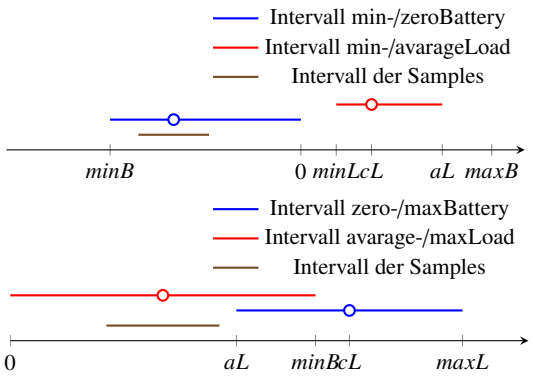


Abb. 6.8 Sampling Modus 3 (Sampling Modus Relativer Gegenwärtiger Verbrauch)

Bei Lastwerten unter der Tageslast wird deshalb das Intervall der minimalen bis durchschnittlichen Last mit der gegenwärtigen Last auf das Intervall der minimalen Beladung (minB) bis zum Standby der Batterie abgebildet, wie in dem oberen Intervall in Abbildung 6.8 dargestellt. Anschließend wird der ermittelte Wert mit einem zufälligen Wert verrechnet. Die Rekombination dieser beiden Werte kann einen weiteren Ausgangspunkt für Evaluierungen bieten. Zur Zeit werden die beiden Faktoren

multipliziert mit einem Faktor addiert. Es könnten aber auch andere Rekombinationen genommen werden oder verschiedene Faktoren ausgewählt werden.

Analog zu den Lastwerten unterhalb der durchschnittlichen Tageslast werden auch die Werte über der durchschnittlichen Tageslast abgebildet, wie in Abbildung 6.8 verdeutlicht.

6.7.2 BHKW

Im diesem Abschnitt wird das Vorgehen für das Sampling des BHKWs dargestellt. Eine Beschreibung des verwendeten BHKW-Modells findet sich in Abschnitt 5.6.

Vorbereitung

Das BHKW Modell besitzt einen großen internen Zustand, welcher in Form von Simulink-Memory Blöcken und Simulink-Data Store Blöcken vorliegt. Damit die für das Sampling verwendete Kopie des Matlabmodells den gleichen Zustand wie das Original besitzt, muss der interne Zustand des Originals vor dem Sampling auf die Kopie übertragen werden. Dazu werden per ADS-Befehle in PLC die Datenbereiche der Memory und Data Store Blöcke des Originals ausgelesen und in die Initialwerte der Memory und Data Store Blöcke der Kopie übertragen. Die Initialwerte sind die Werte, welche ein Block zu Beginn der Simulation annimmt. Ein direktes Beschreiben der eigentlichen Werte der Blöcke war nicht möglich, sondern nur die Initialwerte sind beschreibbar.

Dieses Vorgehen ist somit noch nicht ausreichend, da die Initialwerte nur einmal verwendet werden. Um dieses Problem zu umgehen wurde das Matlabmodell um ein Initflag erweitert, welches bei Aktivierung dafür sorgt, dass die Werte der Blöcke mit ihren Initialwerten überschrieben werden. Vorteil daran ist, dass die Dateninhalte pro Zyklus nur einmal per ADS übertragen werden müssen. Werden beispielsweise 1000 Samples im Zyklus benötigt, kann mittels Initflag nach Erstellung eines Samples der Zustand vor der Erstellung dieses Samples wiederhergestellt werden.

Das Sampling

Wie im Entwurf der Samplingkomponente in Abschnitt 5.7 beschrieben, ist ein schrittweises Sampling sinnvoll, also nach jedem Sampleschritt zu überprüfen, ob dieser valide ist und falls nicht diesen rückgängig zu machen. Dieses Vorgehen ist im BHKW-Modell nicht möglich, weil wie oben beschrieben das Modell einen großen internen Zustand hat. Um einen Schritt rückgängig zu machen, wären bei jedem Schritt ADS-Befehle notwendig, welche den alten Zustand auf die Blöcke des Modelle übertragen. Da ADS-Befehle sehr langsam sind, würden sie die Ausführung des Sampling unnötig verlangsamen. Ein einziger ADS-Befehl kann unter Umstän-

den länger dauern als alle Sampleschritte eines Samples zusammen. Ein weiterer Nachteil ist, dass mittels busy-waiting überprüft werden muss, ob der Befehl bereits fertig ist. Zwar waren die ADS-Befehle für die Vorbereitung des Samplings akzeptabel, da die Vorbereitung nur einmal in einem Sampling geschieht, jedoch kann die Anzahl invalider Schritte beim Sampling hoch sein und somit sind viele ADS-Befehle notwendig.

Deshalb wurde entschieden, das Problem invalider Samplings für das BHKW zu vernachlässigen, da das Hauptaugenmerk unseres Projekts auf das EMS und SF liegt und nicht auf das Sampling der EVS-Komponenten. Um dennoch sinnvolle Samples zu haben, wurde versucht, das Problem invalider Sampling zwar nicht zu lösen aber mit einfachen Methoden zu verringern. Dabei heißt valide hier, dass mindestens eine Stunde lang keine Schaltvorgänge gemacht werden, d.h. vier Zyklen lang keine Änderung notwendig ist. Schaltvorgänge, also Änderungen der Leistung, werden notwendig, wenn die Rücklauftemperatur zu hoch oder zu niedrig ist.

Das Vorgehen des Samplings lässt sich wie folgt beschreiben:

- Initialisiere das Modell mit dem internen Zustand des Originals
- $i := 0$
- Wiederhole bis $i = 96$:
  - Wenn in den letzten vier Zyklen keine Änderung gemacht wurde,...
    - Falls Differenz der beiden vorherigen Rücklauftemperaturen in Zukunft auf zu hohe oder zu niedrige Rücklauftemperatur hindeutet, versuche Leistung[i] dementsprechend anzupassen.
    - Ansonsten setze Leistung[i] zufällig.
  - Simuliere das Modell.
  - Falls Rücklauftemperatur nach Simulation zu hoch, setze Leistung[i] auf 0. Falls Rücklauftemperatur zu niedrig, setze Leistung[i] auf Maximum. (Nachimplementierung von Notfällen des Matlabmodells)
  - $i := i + 1$

Dieses einfache Vorgehen wurde in mehreren Testläufen überprüft. Zwar besitzt es weniger Notabschaltungen und Notanschaltungen als ein Vorgehen ohne Einbezug der vorherigen Rücklauftemperaturen, dennoch ist die Anzahl von Schaltvorgängen für einen echten Gebrauch zu hoch. Für unsere Testzwecke ist es aber ausreichend. Ein typischer Verlauf für die erzeugte elektrische Leistung ist in Abbildung 6.9 zu sehen. Es ist zu beobachten, dass in den meisten Fällen eine Leistung nicht vier Zyklen lang hält. Durch zufällige Wahl einer Leistung werden unter Umständen Leistungen gewählt, die so hoch sind, dass sie im nächsten Zyklus zur Notabschaltung führen.

Dieses Problem könnte vermieden werden, indem im Sampling mehr Wissen über das BHKW einfließt. Dazu müssten jedoch große Teile des Modells für das Sampling nachimplementiert werden, denn das BHKW Modell ist sehr komplex und lässt sich nicht durch einfache Regressionsformeln abbilden. Dieser zusätzliche Aufwand würde aber gerade einen großen Vorteil des Samplings beseitigen. Das Samplingkonzept wurde auch deshalb gewählt, um unnötige Nachimplementierungen von internem Modellverhalten zu vermeiden. Da, wie oben beschrieben, für unsere Ziele ein nicht

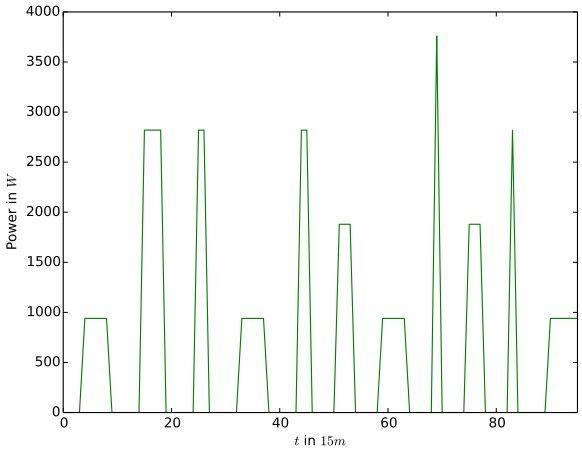


Abb. 6.9 Verlauf der elektrischen Leistung für ein Testsample

realistisches Sampling ausreicht, haben wir uns gegen diesen zusätzlichen Aufwand entschieden.

Alternativen

Um wirklich alle Vorteile von Sampling nutzen zu können, ist es sinnvoll, invalide Schritte rückgängig zu machen. Wie gesagt, ist dies durch den großen internen Zustand des BHKW-Modells nicht möglich. Der interne Zustand lässt sich jedoch vermeiden, indem Memory Blöcke entfernt werden und mit einem Input und Output ersetzt werden. Die Speicherung des internen Zustands erfolgt dann über den CEM, indem der alte Output des letzten Schritts in den Input des nächsten Schritts geleitet wird. Somit hätte das Modell keinen internen Zustand mehr, sondern dieser Zustand wird auf den CEM ausgelagert.

Die Modelle der anderen EVS-Komponenten wurden von uns so konstruiert, dass keine Memory-Blöcke notwendig sind. Dadurch lässt sich für diese Sampling besser realisieren. Um das BHKW Modell ähnlich anzupassen, müsste noch ein analoges Vorgehen für die Data-Store-Blöcke gefunden werden, welche große Arrays speichern können. Dies ist aus Zeitgründen bisher nicht möglich gewesen.

6.7.3 PV-Anlage

Die PV-Anlage ist ein nicht steuerbarer Erzeuger, der abhängig von Sonnenstand und Wetter unterschiedlich viel Leistung produziert. Damit jedoch unser System

die PV-Anlage nicht speziell behandeln muss und der Zugriff analog funktioniert, existiert auch für die PV-Anlage ein CEM mit Samplingkomponente. Die Samplingkomponente erhält als Eingabe Daten über Strahlungswerte und Sonnenstand und berechnet dann die Leistungswerte, die voraussichtlich in den nächsten 24 Stunden erzeugt werden. Dieser Verlauf ist wie ein Sample aufgebaut und die Optimierung kann diesen Tagesplan der PV-Anlage auch als solchen behandeln und in seine Rechnung einfließen lassen.

6.7.4 Steuerbarer Verbraucher - Waschmaschine

Das Sampling der Waschmaschine besteht darin, zufällig einen Startzeitpunkt auszuwählen. Dabei ist vorgegeben, wie oft die Waschmaschine in den nächsten 24 Stunden gestartet werden soll. In Abhängigkeit der Beladung und der Temperatur eines Waschgangs wird dann durch das zugrunde liegende Simulationsmodell der Stromverbrauch für die nächsten 96 15-Minuten-Intervalle in einem Vektor abgespeichert. Die Auswahl eines Startzeitpunkts geschieht über eine Heuristik, die die Nebenbedingungen der Waschmaschine beachtet. Dazu zählen die Anzahl der Waschvorgänge in den nächsten 24 Stunden, die Dauer eines Waschvorgangs sowie die Bedingung, dass die Waschmaschine nur dann gestartet werden kann, wenn sie nicht in Betrieb ist. Die Heuristik führt für jeden der 96 Zeitschritte eine Zufallsfunktion aus, die mit der Wahrscheinlichkeit $\frac{n}{96}$ bestimmt, ob in dem betrachteten Zeitschritt die Waschmaschine einen Waschvorgang startet. Dabei steht n für die Anzahl der Waschvorgänge in den nächsten 24 Stunden. Anschließend prüft die Heuristik, ob durch das Starten bzw. Nichtstarten in einem Zeitschritt eine Bedingung verletzt ist und korrigiert dies gegebenenfalls.

Da lediglich die Startzeitpunkte für die Anzahl der vorgegebenen Waschvorgänge variiert werden, reichen etwa 25 Samples aus, um der Optimierung eine ausreichend große Auswahlmöglichkeit zu bieten. Dies liegt darin begründet, dass ein Waschvorgang zwischen 40 und 120 Minuten, also drei bis acht 15-Minuten-Intervalle dauert. Soll in den nächsten 24 Stunden nur ein Waschvorgang gestartet werden, so kann dieser in jedem der nächsten 96 15-Minuten-Intervalle gestartet werden. Sind allerdings zwölf Waschvorgänge unter Maximallast vorgesehen, so muss in jedem achten Intervall die Waschmaschine gestartet werden. Es ergibt sich dann nur eine Möglichkeit ein Sample zu erstellen. Alle 25 Samples sind in diesem Fall gleich. Für zwei und mehr Waschvorgänge in 24 Stunden ergibt sich eine größere Anzahl an verschieden Samples. Allerdings hat sich nach einigen Tests heraus gestellt, dass bereits 25 Samples für die Optimierung ausreichend sind. Dies liegt u.a. darin begründet, dass die Optimierung nach jedem 15-Minuten-Intervall ein neues Sampling anfordert.

6.7.5 Steuerbarer Verbraucher - Kühlhaus

Ein Kühlhaus bietet nur sehr wenig Raum, Einfluss auf den Energieverbrauch zu nehmen. Je nach Verwendungszweck ist die zu haltende Kühltemperatur als Hard-Constraint zu betrachten. D.h. weder ein Überschreiten noch ein Unterschreiten dieser Zieltemperatur ist gewünscht. Um dennoch ein Sampling anbieten zu können, wird diese Bedingung abgeschwächt. Die tatsächliche Temperatur im Kühlhaus darf die Zieltemperatur unterschreiten, sie aber nicht überschreiten. Dadurch wird es möglich, einen temporären Energieüberschuss zu nutzen, indem das Kühlen des Kühlhauses zeitlich vorgezogen wird.

Das Sampling des Kühlhauses besteht darin, für die nächsten 96 Zeitschritte einen Fahrplan für die Kühltemperatur zu erstellen. Dabei kann die Zieltemperatur unterschritten werden. Allerdings nur soweit, wie es die minimale Kühltemperatur des Kühlhauses erlaubt. Die Zieltemperatur wird dabei nicht überschritten. Ob die Temperatur im Kühlhaus erniedrigt oder erhöht wird, wird durch eine Zufallsfunktion entschieden. Dabei wird in einem 15-Minuten-Zeitintervall die Temperatur mit einer Wahrscheinlichkeit von 10% um ein Grad Celsius entweder erhöht oder erniedrigt. Mit einer Wahrscheinlichkeit von jeweils 5% wird sie um zwei Grad erhöht oder erniedrigt. Das Kühlintervall ist in Abbildung 6.10 verdeutlicht.

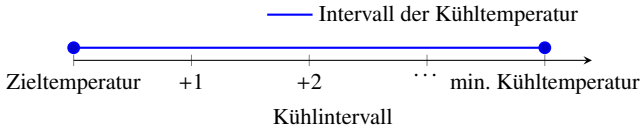


Abb. 6.10 Intervall der Kühltemperatur

Wenn das Intervall der Kühltemperatur außer Acht gelassen wird, gibt es 5^{96} mögliche Fahrpläne. Da das Kühlintervall aber in der Regel sehr klein ist, reichen wenige Samples aus, um der Optimierung ausreichend Spielraum zu lassen. Daher werden bei jedem Aufruf der Sampling-Methode 100 Samples erstellt.

6.8 Komponentenmodelle

Im folgenden Kapitel wird die Realisierung der einzelnen EVS-Komponenten beschrieben.

6.8.1 BHKW

Wie im Kapitel BHKW erwähnt, verwendet das Matlab-Simulink Modell des BHKW M-S-Function-Blöcke. Diese verhindern es, direkt aus dem Modell Code zu generieren und daraus ein TcCom zu erstellen. Es gibt grundsätzliche drei Möglichkeiten, um ein Matlab-Simulink Modell mit S-Functions dennoch zur Codegenerierung nutzen zu können:

- Bereitstellen von tlc-Dateien, die vom Simulink-Coder automatisch beim Kompiliervorgang anstelle der S-Functions verwendet werden.
- Entfernen der S-Functions und Nachempfinden des Verhaltens durch Simulink-Blöcke.
- Entfernen der S-Functions und Nachempfinden des Verhaltens durch M-Functions.

Eine tlc-Datei (Target Language Compiler) wird genutzt, um die Erstellung von Code direkt zu beeinflussen. Im Falle der Codegenerierung aus einer S-Function werden sogenannte Block-Level-tlc-Dateien verwendet. Beim Prozess der Codegenerierung aus einem Matlab-Simulink Modell sucht der Simulink-Coder automatisch nach einer gleichnamigen tlc-Datei, sobald er auf eine S-Function trifft. Ist keine tlc-Datei vorhanden, so wird die Codegenerierung mit einem Fehler beendet.

Mit Hilfe einer tlc-Datei ist die Codegenerierung möglich. Allerdings muss das komplette Verhalten der M-S-Function in einer tlc-Datei neu programmiert werden und zwar auschließlich mit Anweisungen, die der Simulink-Coder ausführen kann. Da die M-S-Functions allerdings sehr umfangreich sind, scheint es zunächst weniger sinnvoll die gesamte Funktionalität neu zu programmieren.

Dieses gilt auch für die zweite Möglichkeit, bei der die S-Functions komplett entfernt werden und ihr Verhalten durch Blöcke, die die Simulink-Bibliothek liefert, nachempfunden werden. Zudem würde das Matlab-Simulink Modell durch zu viele Blöcke sehr unübersichtlich werden und es wäre nötig, diese sinnvoll in Unter- und Überblöcken zu gliedern. Dieses wiederum erfordert ein tiefergehendes Verständnis der Funktionalität, was bisher nicht notwendig ist.

Abhilfe schafft die dritte Möglichkeit. Hierbei werden die S-Functions zwar entfernt, allerdings können große Teile des Quellcodes einfach kopiert werden. Die M-S-Function-Blöcke werden durch M-Function-Blöcke ersetzt, die dieselbe Programmiersprache wie die M-S-Function-Blöcke verwenden. Die M-Function-Blöcke sind zur Codegenerierung geeignet. Allerdings bieten sie nicht dieselbe Funktionalität wie eine S-Function. Dazu zählt beispielsweise das Zustandsvektoren und globale Variablen nicht zwischen den Ausführungsschritten aufrecht erhalten werden können, sondern als Output des Blockes und durch zusätzliche spezielle Speicher-Blöcke gespeichert werden müssen. Außerdem fehlt die Möglichkeit das Verhalten des Blockes durch Callback-Methoden zu verändern.

Trotzdem haben wir uns für die dritte Möglichkeit entschieden. Im Gegensatz zu den ersten beiden Möglichkeiten können wenigstens Teile des Quellcodes der M-S-Functions übernommen werden und es muss nur eine überschaubare Menge an Funktionalität nachempfunden werden. Glücklicherweise sind zudem die Zustände

der verwendeten S-Functions des BHKWs alle diskret, was ein Nachempfinden der Eigenschaften eines solchen Blockes erleichtert.

Im Matlab-Simulink Modell des BHKW mussten drei M-S-Functions durch M-Functions ersetzt werden. Dabei musste der Quellcode an die neue Syntax der M-Function angepasst werden. Dies bestand hauptsächlich in einer anderen Spezifikation der Ein-und Ausgaben des Blockes sowie im Entfernen der Callback-Methoden der M-S-Function. Zudem mussten globale Variablen entfernt werden. Diese wurden in den M-Functions als weitere Ein- und Ausgabe des Blockes definiert und mit Hilfe eines Memory-Blockes zwischen den Ausführungsschritten gespeichert.

Nachdem die M-S-Function-Blöcke durch M-Function-Blöcke ersetzt wurden, ließ sich aus dem Matlab-Simulink Modell ein TcCom generieren. Dieses kann direkt in TwinCAT eingebunden werden. Damit die Ein- und Ausgaben über ADS ansprechbar sind, mussten allerdings noch die Ein- und Ausgaben der Matlab-Simulink Modells durch Input- und Output-Blöcke der Simulink-Bibliothek ersetzt werden. Diese werden durch das TwinCAT Target für Matlab-Simulink automatisch als ADS-Input bzw. Output-Port übersetzt. Außerdem wurde das BHKW Modell so übersetzt, dass innerhalb der TwinCat-Umgebung bzw. über ADS jede interne Variable ausgelesen werden kann. Dies macht es möglich, den exakten Zustand des laufenden BHKWs zu bestimmen, was später u.A. für das Sampling wichtig ist.

Abschließend wurde das generierte TcCom in TwinCat eingebunden und getestet. Dafür wurde ein PLC-Skript (Programmable Logic Controller) angelegt, das auf einfache Weise das Ausführen eines TcCom auf einem Beckhoff Koppler erlaubt. Dabei können die Eingaben direkt in dem PLC-Skript vorgenommen und Ausgaben ausgelesen werden. Außerdem ist es möglich die Ausführung zu starten und zu stoppen. Mit dem PLC-Skript ließ sich nachweisen, dass das generierte TcCom bei gleichen Eingaben die selben Ausgaben wie das Matlab-Simulink Modell lieferte. Das PLC-Skript kann um das Sampling erweitert werden. (Siehe Abschnitt 5.7)

6.8.1.1 Weitere Herausforderungen

Ein weiteres Problem mit dem BHKW Modell war die vorhandene Integration des Gebäudemodells für den Heizkreislauf und des Pufferspeichers in das Matlabmodell. Dadurch ist es zum Beispiel nur schwierig möglich eine weitere Heizquelle in den Heizkreislauf zu integrieren. Um dieses Problem zu lösen, mussten wir deshalb den Wärmebedarf den das Modell als Input erhält im Fall einer zweiten Heizung begrenzen. Daher erhält das BHKW nur den Wärmebedarf, den es theoretisch maximal erfüllen kann. Weitere Heizleistung, welche den Wert den maximalen Leistungswert des BHKW (oder der Heizungen) übersteigt, wird in der Optimierung als zusätzliche Kosten eingerechnet. Die Schwelle, ab welchen Wärmebedarf zusätzliche Heizleistung „importiert“ bzw. bezahlt wird, kann vom Benutzer selbst definiert werden. Die Überlegung dahinter ist, dass ein BHKW für ein großes Gebäude nicht in allen Fällen kosteneffizient ist und es sinnvoller sein kann, eine herkömmliche Heizung neben den BHKW zu betreiben. Die Schwelle kann aber auch auf einen sehr hohen, nicht

erreichbaren Wert gesetzt werden, falls neben dem BHKW keine weitere Heizung vorliegt.

Eine sinnvolle, aber sehr aufwendige Alternative, wäre die Entwicklung eines neuen BHKW-Modells ohne Pufferspeicher und Gebäudemodell, das jedoch die Anbindung von separaten Pufferspeichern und Gebäudemodell erlaubt. Das würde auch die Erweiterung des Systems, um mehrere Heizungen erlauben. Dieses wurde nicht umgesetzt, da der Kern des Projekts nicht die Entwicklung von Matlabmodellen war.

6.8.2 Batterie

In diesem Abschnitt wird die Implementierung eines Batteriemodells erläutert. Die Implementierung einer Batterie mit CCCV Ladeverfahren in Matlab orientiert sich an dem Entwurf eines Batteriemodells in Abschnitt 5.6.3. Entsprechend ist das Modell stark vereinfacht. Um eine spätere Erweiterung des Modells zu ermöglichen, sind bereits als Eingangsparameter Energieeffizient und Energieverlust vorgesehen. Diese werden von dem gegenwärtigen Modell aber nicht berücksichtigt.

In der ersten Version wurde das Modell mit Memory Blöcken für den aktuellen Ladestand der Batterie sowie die vergangene Zeit seit dem letzten Wechsel zwischen Lade- und Entladezyklus umgesetzt. Diese Lösung scheint intuitiv passend zu sein, weil sie den Zustand der Batterie abbildet. Allerdings erschwert die Verwendung von Memory Blöcken das Sampling erheblich, wie in Abschnitt 6.7.2 erläutert. Deshalb sind neben den in Abschnitt 5.6.3 genannten Eingabeparametern noch der aktuelle Ladestand der Batterie sowie die Zeit seit dem letzten Wechsel zwischen Lade- und Entladezyklus hinzugekommen. Die aktuellen Werte werden von dem Modell ausgegeben. Der Customer Energy Manager, wie in Abschnitt 5.7.1 beschrieben, berücksichtigt die entsprechenden Werte und gibt sie für den nächsten Simulationsschritt dem Modell als Eingabeparameter wieder mit. Auf diese Art und Weise ist es einfach möglich einen Zustand der Batterie für das in Abschnitt 6.7.1 beschriebene Sampling wieder herzustellen.

Das Modell berechnet anhand des gegenwärtigen Ladestandes, wie viel Eingangsleistung möglich ist. Deshalb kann der Kern des Modells mit der möglichen Eingangsleistung über dem Ladestand dargestellt werden, wie in Abbildung 6.11 mit den Parametern aus Abbildung 6.12 visualisiert. Bis zu einem Ladestand der Batterie von bis zu etwa 1,2 kWh kann die Batterie mit maximaler Eingangsleistung geladen werden. Ab einem Ladestand von ca. 1,2 kWh wird vom Konstantstrom- aufs Konstantspannungs-Ladeverfahren umgestellt. Damit nimmt mit zunehmenden Ladestand die mögliche Eingangsleistung immer weiter ab. Intern wird in dem Modell über die in Abschnitt 5.6.3 vorgestellten Formeln zunächst der aktuelle Zeitpunkt t für den Ladestand der Batterie bestimmt. Über diesen wird anschließend berechnet, wie viel Eingangsleistung maximal möglich ist.

Das Verhalten des Batteriemodells kann am besten anhand eines Simulationslaufs erläutert werden. Im Folgenden wird ein Simulationslauf mit einigen besonderen

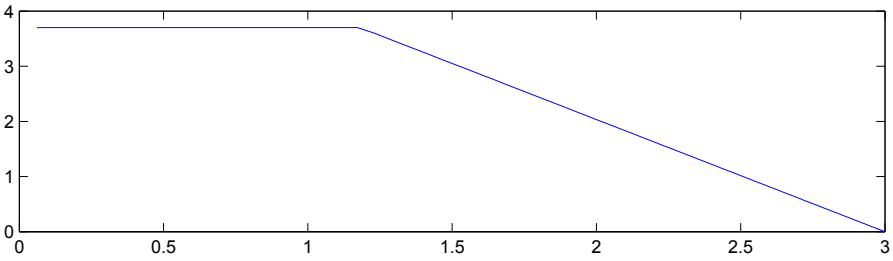


Abb. 6.11 Mögliche Eingangsleistung in kWh über den Ladestand in kWh

Ereignissen vorgestellt. In Abbildung 6.13 ist die Eingangsleistung der Batterie visualisiert. Der blaue Graph ist die geforderte Eingangsleistung, während im roten Graphen die tatsächliche Eingangsleistung dargestellt wird. Falls zu viel oder zu wenig Eingangsleistung von der Batterie gefordert wird, so gibt sie als tatsächliche Eingangsleistung nur so viel ab, wie es ihr technisch möglich ist. Diese ist im roten Graphen visualisiert. Damit eine Diskrepanz zwischen geforderter und tatsächlicher Leistung einfacher festgestellt werden kann, gibt die Batterie auch zurück, ob die geforderte Leistung eine valide Eingabe war. Dieses Anpassen der geforderten Energie ermöglicht es der Batterie immer in einem konsistenten Zustand zu sein. Der Verlauf dieses Ausgabeparameters ist in Abbildung 6.14 dargestellt. Diese beiden Werte erleichtern das in Abschnitt 6.7.1 dargestellte Sampling. Um einen Einblick in den Zustand der Batterie zu ermöglichen, ist der Ladestand in Abbildung 6.15 und die Zeit zwischen Wechsel des Lade-/Entladezyklus in Abbildung 6.16 abgebildet. Die Parameter des Simulationslaufs finden sich in Abbildung 6.12.

Zeitspanne des Konstantspannungs - Ladeverfahren:	1800 Sekunden
Zeit zwischen Wechsel des Lade-/Entladezyklus:	1800 Sekunden
Eingangsleistung:	Abbildung 6.13
Maximale Eingangsleistung:	3,7 kWh
Minimale Eingangsleistung:	-3,7 kWh
Maximale Kapazität:	3 kWh
Intervallgröße:	60 Sekunden

Abb. 6.12 Parameter des Simulationslaufs

Im Verlauf des Ladestands der Batterie in Abbildung 6.15 kann der in Abschnitt 5.6.3 beschriebene Verlauf einer Batterie mit CCCV Ladestrategie ab Minute 150 sehr gut nachvollzogen werden. Zunächst lädt die Batterie mit konstantem Strom und ab in etwa Minute 160 findet der Wechsel zum Konstantspannungs - Ladeverfahren statt.

Der Verlauf der Simulation beinhaltet einige besondere Ereignisse. Dies beginnt beim Minute 25. Hier wird ein Wechsel vom Laden zum Entladen der Batterie

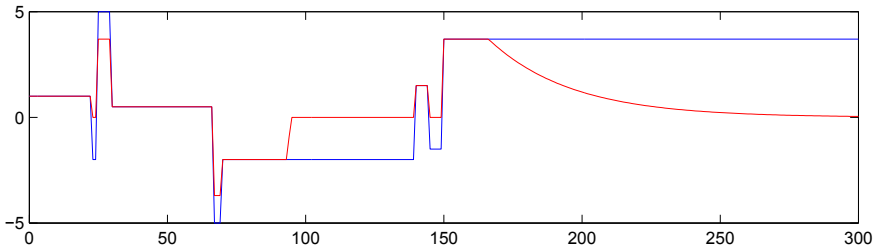


Abb. 6.13 geforderte Eingangsleistung (blau) tatsächliche Eingangsleistung (rot) in kWh über die Zeit

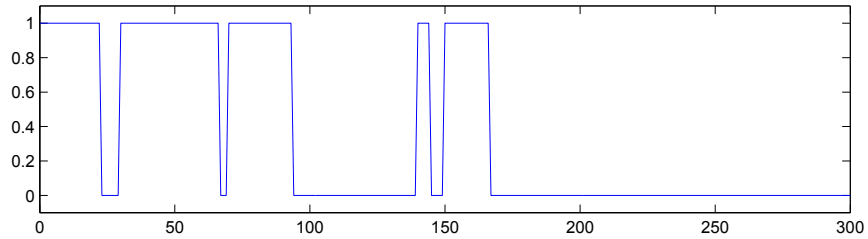


Abb. 6.14 Validierung der Eingangsleistung in Wahrheitswert über die Zeit

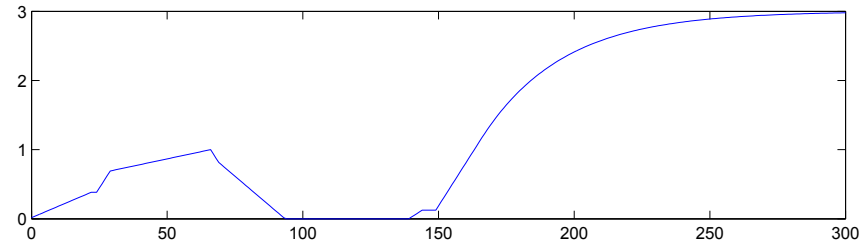


Abb. 6.15 Ladestand der Batterie in kW über die Zeit

gefordert, obwohl die Zeit für einen Wechsel zwischen diesen Phasen noch nicht erreicht ist. Dies führt zum einen dazu, dass die Validierung in Abbildung 6.14 ein negatives Ergebnis liefert und zum anderen wird als tatsächliche Eingangsleistung in Abbildung 6.13 null ausgegeben. Dieser Stillstand der Batterie kann auch sehr gut im Verlauf des Ladestandes in Abbildung 6.15 beobachtet werden.

Nach diesem unerlaubten Versuch zu früh Leistung von der Batterie zu bekommen, zeigt sich ein weiteres interessantes Verhalten. Es soll mehr Energie eingespeist werden, als technisch möglich ist. Auch dieser Fall führt in dem vorliegenden Modell zu einer negativen Validierung und die tatsächliche Leistung wird den technischen

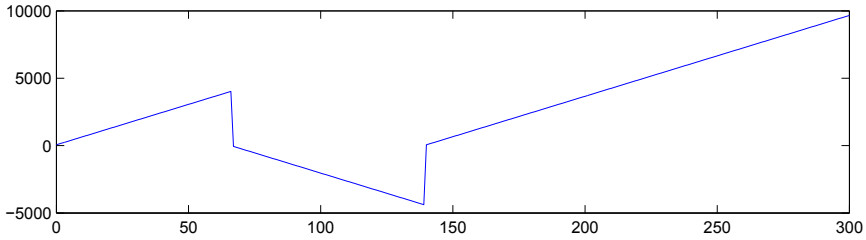


Abb. 6.16 Zeit seit dem letzten Wechsel zwischen Lade-/Entladezyklus in Sekunden über die Zeit

Möglichkeiten angepasst. In diesem Fall wird die Leistung auf die maximale Leistung beschränkt.

Das umgekehrte Verhalten kann von Minute 67 bis 70 in Abbildung 6.13 beobachtet werden. Hier wird versucht mehr Leistung von der Batterie zu fordern, als technisch möglich ist. Die Folge ist wieder eine negative Validierung sowie eine Anpassung der tatsächlichen Leistung an die technischen Möglichkeiten.

Ein weiterer Fehlerfall im Zusammenhang mit der Leistungsentnahme aus der Batterie kann ab ungefähr Minute 90 beobachtet werden. Die Energie soll Leistung abgeben, allerdings hat sie keine Leistungsreserven mehr. Deshalb erfolgt in diesem Fall eine negative Validierung und die tatsächliche Eingangsleistung wird auf null erhöht.

In Minute 140 kann eine Designentscheidung des Modells beobachtet werden. Die Zyklen ohne Lade-/Entladevorgang werden jeweils zu der Zeit bis zu einem Wechsel hinzugerechnet, wie in Abbildung 6.16 sehr schön zu sehen ist. Deshalb ist es möglich die Batterie zu laden, obwohl noch keine 30 Minuten entladen wurde. Allerdings ist die Zeit des Stillstandes und die Zeit der Leistungsentnahme größer als die Zeitspanne, die vor einem Wechsel gefordert wird. Ohne dieses Verhalten wäre die Batterie in diesem Fall in einen Deadlock gekommen und hätte nie wieder Energie aufnehmen können.

Direkt nach dem Wechsel zum Laden wird versucht in dem Simulationsdurchlauf versucht wieder Energie zu entnehmen. Allerdings hat gerade erst ein Wechsel stattgefunden, weshalb die Validierung fehlschlägt und die Batterie in den Stillstand gefahren wird, wie bereits bei Minute 25 erläutert.

Abschließend zeigt sich ab Minute 160, dass die Validierung auch fehlschlägt, wenn die gewünschte Einspeisung zwar innerhalb der technischen Möglichkeiten der Eingangsleistung liegt, die Batterie aber aufgrund des Wechsels zum Konstantspannungs - Ladeverfahren weniger Energie aufnehmen kann. Auch hier schlägt die Validierung fehl und die tatsächlich mögliche Energie wird an das Ladeverfahren angepasst ausgegeben.

6.8.3 PV-Anlage

Wie beim BHKW verhindern auch bei der PV-Anlage drei M-S-Function-Blöcke die Generierung von Code aus dem Matlab-Simulink Modell. Auch hier wurden diese durch M-Function-Blöcke ersetzt. Analog zum BHKW wurde dabei der Quellcode an die neue Syntax angepasst und Callback-Methoden entfernt. Zusätzlich dazu tauchten noch spezielle Anweisung an den Workspace von Matlab auf, die ebenfalls entfernt wurden. Im Gegensatz zum Matlab-Simulink Modell des BHKWs gab es keine globalen Variablen in den M-S-Function-Blöcke, die einen Memory-Block im Modell der PV-Anlage bedingt hätten.

Ohne die M-S-Function-Blöcke ließ sich aus dem Modell ein TcCom erstellen. Auch wurden bei der PV-Anlage die In- und Output-Blöcke aus der Simulink-Bibliothek ergänzt, um Ein- und Ausgaben über ADS ansprechbar zu machen.

Im gesamten Modell der PV-Anlage gab es keine Blöcke, die zum Speichern von Information (Data Store- bzw. memory-Blöcke) dienten. Deshalb war es auch nicht notwendig, direkten ADS-Zugriff auf die internen Variablen herzustellen. Die gesamte Berechnung hängt in jedem Zeitschritt nur von den Eingaben ab. Dies macht es auch einfacher, das Sampling für die PV-Anlage zu implementieren.

Analog zum BHKW wurde die PV-Anlage in TwinCat eingebunden und mit Hilfe eines PLC-Skripts getestet.

6.8.3.1 Weitere Herausforderungen

Die PV-Anlage benötigt als Eingabewerte neben Sonnenständen auch Werte über das Umgebungsalbedo, also dem Rückstrahlvermögen der Umgebung. Da die Simulation der näheren Umgebung der PV-Anlage (z.B. Wandmaterial, Schnee etc.) für unsere Aufgabe zu aufwendig ist, haben wir uns dazu entschieden für den Albedowert einen konstanten Wert anzunehmen, den der Benutzer für die Komponenten definieren kann.

Weiterhin wurden Abfragen aus dem PV-Modell ausgebaut, die *NotANumber* (*NaN*) erzeugen. In Matlab führen diese nicht zum Absturz, sondern können mit Abfragen wie *isNaN* abgefangen werden. Dies ist im generierten C++ Code nicht mehr möglich. Es ist daher wichtig, auf Fehler zu achten, die durch die Konvertierung des Matlab-Simulink-Modells zu C++-Code entstehen.

6.8.4 Steuerbarer Verbraucher - Waschmaschine

Der steuerbare Verbraucher wurde wie im Entwurf geschildert als Matlab-Simulink Modell implementiert. Matlab-Simulink wurde als Entwicklungsumgebung gewählt, da bisher alle Komponentenmodelle mit dieser entwickelt wurden. Analog zum BHKW und zur PV-Anlage ist das Verhalten des steuerbaren Verbrauchers in einer M-Function verfügbar. Der interne Zustand ist wie beim BHKW in den soge-

nannten Memory-Blöcken gespeichert. Dieser besteht aus zwei Variablen, die den aktuellen Fortschritt des Waschvorgangs und die Länge eines Waschvorgangs in 15-min Zeitschritten speichern sowie einer Variable, in der gespeichert ist, ob die Waschmaschine bereits gestartet wurde.

Der Stromverbrauch ist wie im Entwurf festgelegt, nicht in jedem Zeitschritt gleich. Stattdessen steigt er bis zur Hälfte der Zeitschritte eines Waschvorgangs an und fällt danach wieder ab.

Auch der steuerbare Verbraucher ließ sich erfolgreich in ein TcCom übersetzen. Anschließend wurde das TcCom in TwinCat eingebunden und mit Hilfe eines PLC-Skripts getestet.

6.8.5 Zweiter steuerbarer Verbraucher - Kühlhaus

Auch der zweite steuerbare Verbraucher wurde als Matlab-Simulink Modell implementiert. Dabei wurde die in Abschnitt 5.6.5 entworfene Funktionalität in einer M-Function realisiert. Insgesamt gibt es 13 Input-Blöcke, die sämtliche Parameter des Kühlhauses bereitstellen. Eine Darstellung dieser Parameter findet sich in Tabelle 6.1. Die Formelzeichen entsprechen denen der mathematischen Modellierung in Abschnitt 5.6.5.2. Der Wertebereich gibt einen sinnvollen Wertebereich für ein Kühlhaus an. Alle Eingaben sind Fließpunktzahlen.

Zudem wird die aktuelle Raumtemperatur und die Information, ob der erste Simulationsschritt ausgeführt wird, jeweils in einem Memory-Block gespeichert. Damit erhält die M-Function 15 Eingabewerte. Diese bestimmen den einzigen Ausgabewert des Matlab-Simulink-Modells - den Verbrauch der elektrischen Energie. Dafür werden zunächst alle Eingaben, die in der Einheit Grad Celsius gegeben sind, in Kelvin umgerechnet. Die Dichte von Wasser und Luft sowie die spezifische Wärmekapazität von Luft sind als konstanten angegeben. Für den ersten Simulationsschritt wird die Raumtemperatur auf die initiale Raumtemperatur gesetzt. Anschließend wird die Kühlleistung über die Formeln aus Abschnitt 5.6.5.2 bestimmt, in Kilowatt umgerechnet und ausgegeben.

6.9 Optimierung

Dieses Kapitel beschreibt die Umsetzung der Optimierung. Die Grundsätzlichen Entscheidungen dazu sind bereits in Abschnitt 5.10 beschrieben, hier werden lediglich die implementierungsspezifischen Aspekte erklärt. Abschnitt 6.9.1 beschreibt den Ablauf der Software. Abschnitt 6.9.2 gibt Auskunft über die Schnittstellen der Optimierung. Abschnitt 6.9.3 beschreibt die Aufteilung der Optimierung in kleinere funktionale Blöcke. Abschnitt 6.9.4 erläutert wie die Fitness zustande kommt und Abschnitt 6.9.5 beschreibt den Algorithmus zur Findung eines guten Individu-

Name	Formelzeichen	Einheit	Wertebereich	Beschreibung
height	h	Meter	$2-5$	Höhe des Kühlhauses
width	b	Meter	$1-20$	Breite des Kühlhauses
length	l	Meter	$1-20$	Länge des Kühlhauses
enviroment temperature	T_{envi}	Grad Celsius	$0-35$	Umgebungs-temperatur
target temperature	T_t	Grad Celsius	$-15--20$	Kühltemperatur
thermal conductivity	λ	$W/(m*K)$	$0,001-0,1$	Wärmeleitfähig-keit der Wände
thickness	s	Meter	$0.01-1$	Dicke der Wände
initial room temperatur	T_{init}	Grad Celsius	$-20-35$	Anfangs-temperatur
filling level	f	%	$0-1$	Füllstand
temperature tolerance	T_{tol}	Kelvin	$1-10$	Toleranz des Zweipunktreg-lers
min cooling temperature	T_{min}	Grad Celsius	$-20--30$	minimale Kühltemperatur
max cooling efficiency	P	kWh	$0.5-5$	Kühlleistung
efficiency	η	%	$0-1$	Wirkungsgrad

Tabelle 6.1 Parameter des zweiten steuerbaren Verbrauchers

ums. Zuletzt enthält Abschnitt 6.9.6 Informationen über das Echtzeitverhalten der Optimierung.

6.9.1 Ablauf

Bevor die Optimierung arbeiten kann, muss sie mit den Daten des Szenarios und des Runs versorgt werden, um die verwendeten EVS-Komponenten ansprechen zu können. Danach wird die Optimierung in Zyklen betrieben. In jedem Zyklus wird eine Menge von Samples von jeder EVS-Komponente geholt und eine gute Kombination wird gesucht. Am Ende des Zyklus wird den Komponenten mitgeteilt welches der Samples sie jeweils verfolgen sollen.

6.9.2 Schnittstellen

Die Schnittstellen der Optimierung umfassen mehrere Ein- und Ausgaben. Bei der Initiierung übergibt der EMS-Controller die Szenariodaten an die Optimierung. Die-

se enthält Daten über die EVS-Komponenten und die Prognose der thermischen und elektrischen Grundlast. Ebenfalls bei der Initiierung werden die Rundaten übertragen. Hier sind folgende Daten zu finden:

- An- und Verkaufspreis für Strom
- Kosten zur Generierung fehlender Wärme
- Kosten für das Betreiben des BHKW
- die Zeit der Simulation
- der Seed
- den Grenzwert der Wärmeproduktion des BHKW

Außerdem werden in der Datenbank die benötigten Semantiken herausgesucht und weitere Semantiken ergänzt. Während das System läuft, d.h. Komponenten werden aktiv gesteuert, werden zu Beginn einer Optimierung Samples mittels der Komponentenkommunikation geholt. Während der Optimierung wird geloggt sobald eine neue beste Fitness gefunden wird und in welcher Generation dies geschieht. Am Ende der Optimierung wird die Fitness der gewählten Samples als Summe und für einzelne Intervalle in die RunData eingetragen und durch den Controller an die Datenbank übertragen. Die Auswahl der Samples wird im Log festgehalten und mittels Komponentenkommunikation an die EVS-Komponenten übertragen. Das Starten und außerplanmäßige Stoppen der Optimierung erfolgt durch den EMS-Controller. Die Ausgaben an das Log erfolgen mittels des Loggers.

6.9.3 Komponenten der Optimierung

Die Komponenten der Optimierung und deren wichtigste Funktionen sind in Abbildung 6.17 dargestellt. Sie werden im einzelnen in den Folgesektionen beschrieben.

6.9.3.1 Optimizationcomponent

Die Optimizationcomponent ist der Kern der Optimierung. Sie ist die Schnittstelle zur Optimierung für andere Komponenten und enthält den grundsätzlichen Ablauf, also das Vergleichen und Auswählen von Lösungen in der gegebenen Zeit und sowohl das Lesen als auch das Schreiben der Daten an Komponenten außerhalb der Optimierung. Außerdem dient sie als Verbindung der an der Optimierung beteiligten Module und erlaubt somit deren Kooperation. Die für die Optimierung erforderlichen Daten werden ebenfalls in der Optimizationcomponent gespeichert, dies umfasst die Samples und Daten über das Szenario und den Run.
Die Sampledaten liegen als SampleDataList vor und können von den anderen Optimierungskomponenten mittels getValue abgerufen werden. Der emsCompMan ist das ComponentenManagerInterface zur Kommunikation mit den EVS-Komponenten und dient somit dem Auslesen von Samples und Steuern der EVS-Komponenten.

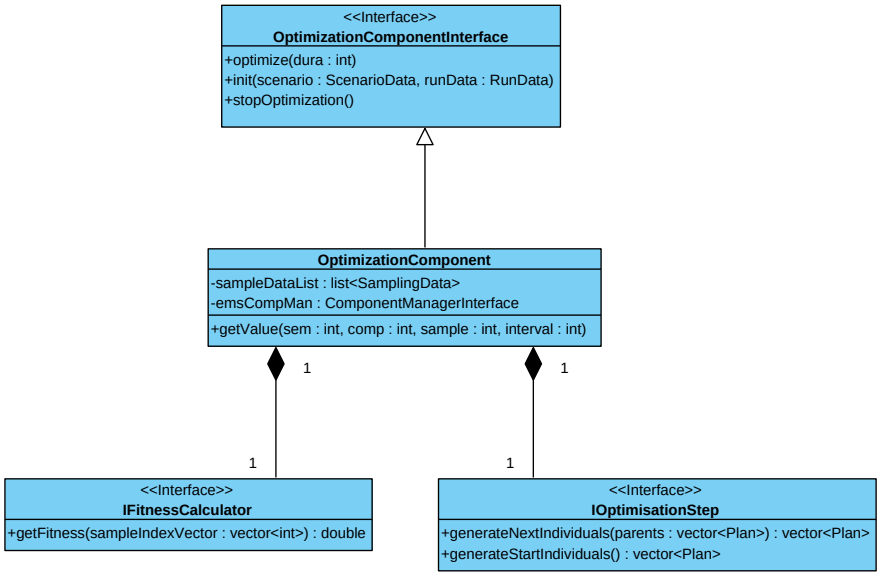


Abb. 6.17 Module der Optimierung

Steuerbar ist die Optimierung mittels der im OptimizationComponentInterface angegebenen Methoden. Sie wird durch init(...) mit den Sample- und Rundaten initialisiert und durch optimize(...) wird für die angegebene Dauer ein Optimierungszyklus durchgeführt. Die Methode stopOptimization() bietet die Möglichkeit die Optimierung vorzeitig abzubrechen.

6.9.3.2 IFitnesscalculator

Die Aufgabe des IFitnessCalculator ist das Berechnen einer Fitness für eine gegebene Auswahl von Samples. Dazu dient die Funktion getFitness(...), welche die Fitness als double ausgibt. Durch den Einsatz verschiedener IFitnessCalculator Module können bei der Optimierung unterschiedliche Ziele verfolgt werden. Es wurde eine Profitorientierte Variante umgesetzt, es sind aber auch eine Eigenverbrauchsoptimierung oder andere Ziele denkbar.

6.9.3.3 IOptimisationstep

Das Modul IOptimisationstep dient dazu Teilschritte der Optimierung im Bezug auf die Erzeugung der Individuen zu erledigen. Die Funktion generateStartIndividuals() erzeugt eine Menge von Individuen zum Start der Optimierung, bei der Beispielhaft umgesetzten Variante werden hier zufällig Samples kombiniert bis 15 Kombina-

tionen erstellt wurden. Diese werden danach von der Funktion generateNextIndividuals(...) verwendet, die bei Aufruf aus einer gegebenen Generation die nächste Generation erstellt. Bei der Umsetzung werden aus den 15 Eltern 100 Kinder erstellt und aus diesen Kindern die 15 besten als nächste Generation verwendet.

6.9.4 Fitnessberechnung

Die Fitness unserer Individuen entspricht den Einnahmen bzw. Ausgaben die durch die EVS-Komponenten und die Gebäudegrundlast in 24 Stunden entstehen. Das heißt je höher die Fitness, desto mehr Geld sollte bei den gewählten Samples eingenommen werden. Diese wird berechnet indem die Einzelwerte der Komponenten und der Grundlast für je 15 Minuten addiert werden. Dazu werden für jedes 15 Minuten Intervall jeweils die elektrischen Ausgangswerte für jede Komponente aus dem jeweiligen Sample gelesen und zusammen mit der Grundlast addiert. Je nachdem ob das Ergebnis positiv oder negativ ist, wird diese elektrische Bilanz mit dem Ankaufs- oder Verkaufspreis multipliziert und zur Fitness addiert. Außerdem wird die thermische Grundlast mit der maximal Leistung des BHKW verglichen und falls die Grundlast größer ist, wird die Differenz mit einem Preis multipliziert von der Fitness abgezogen. Die tatsächliche Produktion des BHKW kann aufgrund des Modellaufbaus nicht verwendet werden. Zusätzlich werden die Kosten für den Betrieb des BHKW anhand der produzierten Wärmeenergie bestimmt und ebenfalls zur Fitness hinzugefügt. Diese Fitnesswerte für alle 96 Intervalle eines Tages werden aufsummiert und ergeben die Fitness für die 24 Stunden Prognose der gewählten Samples.

6.9.5 Algorithmus

Für die Optimierung wird ein genetischer Algorithmus verwendet,welcher wie zuvor beschrieben in IOptimisationStep untergebracht ist. Unsere Individuen haben eine Größe gleich der Anzahl der EVS-Komponenten und enthalten den Index der Samples, welche für das jeweilige Individuum verwendet werden. Wir verwendet eine Mutationsrate von 1/N. Die Rekombinationsstrategie nimmt jede Gerade Stelle in der Folge der Indizes von einem Elternteil und jede andere von dem anderen Elternteil. Die Anzahl der Eltern entspricht 15 und die Anzahl der Kinder ist 100, wobei die Eltern der vorhergehenden Generation nicht in die Auswahl der Eltern der nächsten Generation hinzugezogen werden, dies wird als (15,100) bezeichnet. Das zum Ende gewählte Individuum entspricht dem besten Plan der während der gesamten Optimierung gefunden wurde. Die konkreten Parameter der Optimierung werden noch in der Evaluierung weiter untersucht und ggf. geändert.

6.9.6 Echtzeitverhalten

Der Optimierung steht nur eine begrenzte Zeit zur Erledigung seiner Aufgabe zur Verfügung, deshalb wird nach der Generierung jeder neuen Generation die zur Verfügung stehende Zeit, welche bei Beginn der Optimierung übergeben wird, mit der Zeit zum Start der Optimierung und der aktuellen Zeit verglichen. Wird bei diesem Vergleich festgestellt, dass zu wenig Zeit für eine weitere Generation verbleibt, so wird die Optimierung beendet und das Ergebnis übertragen. Es wird die durchschnittliche Zeit zur Berechnung einer Generation zur Laufzeit ermittelt, um diesen Vergleich durchführen zu können.

6.10 Simulationsframework

Im folgenden Abschnitt wird die Realisierung des Simulationsframeworks (SF) gemäß dem Entwurf in Abschnitt 5.2.3 erläutert. Das SF ist eines der drei Systeme, die im Rahmen des Projekts entwickelt worden sind. Es ist als anwendungsneutrales Programm implementiert, für das keine Benutzereingaben oder -interaktionen benötigt werden. Das SF simuliert das konfigurierte Szenario bestehend aus dem Gebäude, der zugehörigen steuerbaren EVS-Komponenten und den Umgebungsparametern. Die Konfiguration des Szenarios findet in der Steuerungsebene statt. Die folgenden Paragraphen enthalten Informationen über Funktionsweise und Implementierungsdetails des jeweiligen Moduls.

SIMCommunication

Die *SIMCommunication* ist die Schnittstelle zur Steuerungsebene. Im *Konstruktor* der *SIMCommunication* wird der Kommunikationskanal zur Steuerungsebene hergestellt und die Objekte für *Clock* und *SIMController* erzeugt. Zur Initialisierung werden die IDs für das Szenario und den zugehörigen Run an die *SIMCommunication* übergeben, die diese Daten an den *SIMController* weiterleitet.

Clock

Die *Clock* ist der Proxy-Zeitgeber für das gesamte Simulationsframework und ist als *Observer* des Zeitgebers für das Energiemanagementsystem implementiert. Die *Clock* sendet das Taktsignal für jeden *Cycle* an den *SIMController*. Die Realisierung des Zeitgebers ist in Abschnitt 6.4 ausführlich dokumentiert.

SIMController

Der *SIMController* übernimmt die zentrale Steuerungsfunktion des Simulationsframeworks. Während der Initialisierung erzeugt er den *ModelManager* und die *Fieldcommunication*. Mittels *Data Access Layer* wird das zu simulierende Szenario für die von der *SIMCommunication* übergebenen ID geladen. Mit dem Szenario und den zugehörigen Rundaten wird der *ModelManager* initialisiert. Aus dem Szenario wird die *ComponentList* erzeugt, die alle im Szenario integrierten EVS-Komponenten enthält. Diese wird zur Initialisierung der *Fieldcommunication* benötigt. Danach werden die *Flags* der EVS-Komponenten auf "1"gesetzt, um zu signalisieren, dass die Initialisierung abgeschlossen ist. Mit Erhalt eines Taktsignals der *Clock* sendet der *SIMController* alle benötigten Daten, die der *ModelManager* für den jeweiligen *Cycle* bereithält, an die EVS-Komponenten. Zum Ende eines jeden *Cycles* werden die *Flags* wiederum auf "1" gesetzt.

SIMModel

Das *SIMModel* integriert den *ModelManager*, der die geforderten Daten an den *SIMController* liefert und nach dem Konzept in Abschnitt 5.8 realisiert ist. Während der Initialisierung verrauscht er die Daten, die mit einem Kürzel *Distorted* gekennzeichnet sind. Bei diesen Daten handelt es sich um Daten, die als Prognosewerte gespeichert sind und daher zu realen Werten verrauscht werden müssen. Für jeden *Cycle* liefert der *ModelManager* für eine *SemanticID* den entsprechenden Wert.

Fieldcommunication

Die *Fieldcommunication* dient als Schnittstelle zwischen *SIMController* und den EVS-Komponenten auf den Kopplern. Für Simulationsframework und Energiemanagementsystem wird die gleiche Implementierung verwendet. Mit jedem Cycle werden die für die EVS-Komponenten benötigten Inputdaten mittels des *SIMControllers* abgerufen und übergeben.

Data Access Layer

Der *Data Access Layer* ist die Schnittstelle zwischen Simulationsframework und Datenbank. Über den *Data Access Layer* werden benötigte Daten abgerufen und generierte Daten in die Datenbank geschrieben. Da für das Gesamtsystem eine Datenbank verwendet wird, wird für jedes System die gleiche Schnittstelle verwendet.

6.11 GUI

Für die grafische Benutzeroberfläche (engl. GUI) wurden mehrere Frameworks getestet. Es wurde sich für das QT-Framework [57] entschieden, da dieses noch weiterentwickelt wird und umfangreiche Werkzeuge zum Designen und entwickeln von grafischen Benutzeroberflächen bereit stellt.

Die GUI ist nach dem Model-View-Controller Konzept (kurz MVC) aufgebaut. Die Datenbank ist unser Model und stellt die Daten bereit. Jede GUI hat einen eigenen Controller, der der GUI die benötigten Daten und Steuerfunktionen bereitstellt. Durch diese strikte Trennung ist es ohne weiteres möglich die grafische Oberfläche auszutauschen. Das gewählte Framework, in unserem Fall QT, kann einfach durch ein anderes ersetzt werden.

Wie im Abschnitt 5.11 schon erwähnt, ist die GUI in zwei Oberflächen aufgeteilt: der Steuerungs-GUI und der Evaluation-GUI. Die Steuerungs-GUI dient zum Starten, Stoppen, Konfigurieren und zur Beobachtung des Systems im Betrieb, während die Evaluation-GUI zur Evaluation des Systems dient. Die beiden Benutzeroberflächen können unabhängig voneinander ausgeführt werden.

6.11.1 Steuerungs-GUI

Der Controller der Steuerungs-GUI bekommt Daten aus der Datenbank (Struktur der Datenbank in Abschnitt 5.13) und sendet Steuerdaten (Systemkommunikation Abschnitt 5.9) an die anderen Systeme (EMS, SF). Steuersignale werden an das EMS gesendet um beispielsweise das System zu starten. Ist das System gestartet, so werden Rundaten in die Datenbank geschrieben. Der Controller der Steuerungs-GUI erkennt anhand des Zyklus-Signals, welches die Systeme untereinander synchronisiert, dass neue Daten vorhanden sind und fragt diese aus der Datenbank ab. Diese Daten werden an die GUI weitergegeben und dort aufbereitet, sodass sie in den in Abschnitt 5.11 erläuterten Tabellen angezeigt werden können.

6.11.2 Evaluation View

Um eine Offline-Evaluation zu ermöglichen, wurde die *Evaluation View* mit der Idee erstellt, von EMS, SF und Steuerungsebene unabhängig zu sein. Hierfür ist eine eigene Verbindung zur Datenbank erforderlich, um an die benötigten Daten eines Runs zu kommen. Ziel dieses Programms ist, verschiedene Rundaten aufzulisten, sie in einem Graph darzustellen, tabellarisch aufzulisten, sowie den Export dieser Daten zu ermöglichen. Dafür wurden drei unterschiedliche Ansichten zur Anzeige in Tabs implementiert. Zunächst können aus einer Baumstruktur (siehe Abschnitt 6.11.2.1) die zu einem Szenario gehörenden Rundaten ausgewählt werden. Diese Daten werden

dann in Tabellen- (siehe Abschnitt 6.11.2.2) und Plotform (siehe Abschnitt 6.11.2.3) dargestellt.

6.11.2.1 Tree View

Die Tree View (deutsch: Baumansicht) ist das Hauptsteuerelement der graphischen Benutzeroberfläche der Evaluation View. Sie ermöglicht die Auswahl simulierter Szenarien und stellt deren Daten in einer hierarchisch gegliederten Liste dar. Der Baum besteht dabei aus mehreren ineinander geschachtelten Knoten, die wahlweise per Mausklick auf- oder zugeklappt werden können.

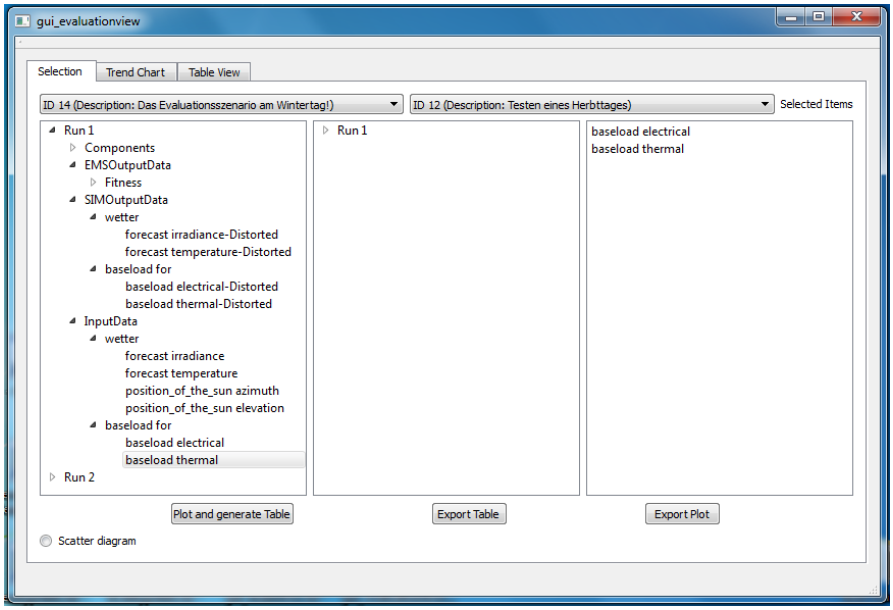


Abb. 6.18 Tree View der Evaluation View

In Abbildung 6.18 ist das Fenster der Tree View abgebildet. Die Tree View bietet zwei Auswahlfenster zur Selektion von jeweils einem Szenario. Dies bietet die Möglichkeit zwei Szenarien gegenübergestellt auszuwerten. Die Szenarien werden direkt mittels Schnittstelle zur Datenbank geladen, sodass keine Abhängigkeiten zu anderen Systemen existieren. Die oberste Hierarchieebene eines Szenarios bilden die zum Szenario zugehörigen Runs. In der Ebene darunter werden die Rundaten abgebildet. Dazu gehören die für das Szenario konfigurierten EVS-Komponenten und Inputdaten (Grundlastprognose, Wetterprognose), die Ausgangsdaten der Optimierung sowie die Ausgangsdaten der Simulation. Die dritte Hierarchieebene bilden

die selektierbaren Attribute beziehungsweise Items. Die Items enthalten Daten, mit denen sie eindeutig für das entsprechende Szenario und den zugehörigen Run identifizierbar sind. Sie können per Doppelklick in das Fenster für selektierte Items geladen und auf diese Art auch wieder aus der Liste gelöscht werden. Befindet sich mindestens ein Item in der Liste für selektierte Items, werden die Buttons zum Erzeugen des Plots und Erstellen der tabellarischen Ansicht, zum Exportieren der Tabelle und zum Exportieren des Plots aktiviert. Weitere Details zur Tabellenansicht und der Plot-Funktion sind in den folgenden Abschnitten beschrieben.

6.11.2.2 Tabellenansicht

Die Tabellenansicht, zu sehen in Abbildung 6.19, nutzt die gleichen Daten, die auch in dem Plot (siehe Abschnitt 6.11.2.3) visualisiert werden. Dabei handelt es sich um die Daten eines Durchlaufes von den zuvor in der Treeview ausgewählten Items. Die Tabelle wird zeilenweise aufgebaut. Eine Zeile besteht aus dem Namen des ausgewählten Items und anschließend den Datensätzen in zeitlich sortierte Reihenfolge. Die verschiedenen Zeilen werden in der Reihenfolge aufgebaut, wie die Datenbank die Daten ausgegeben hat. Fehlen in einer Datenreihe Werte, werden die Felder nicht beschrieben. Dies führt dazu, dass die Anzahl der Spalten von der Länge des Datensatzes mit den meisten Werten abhängt.

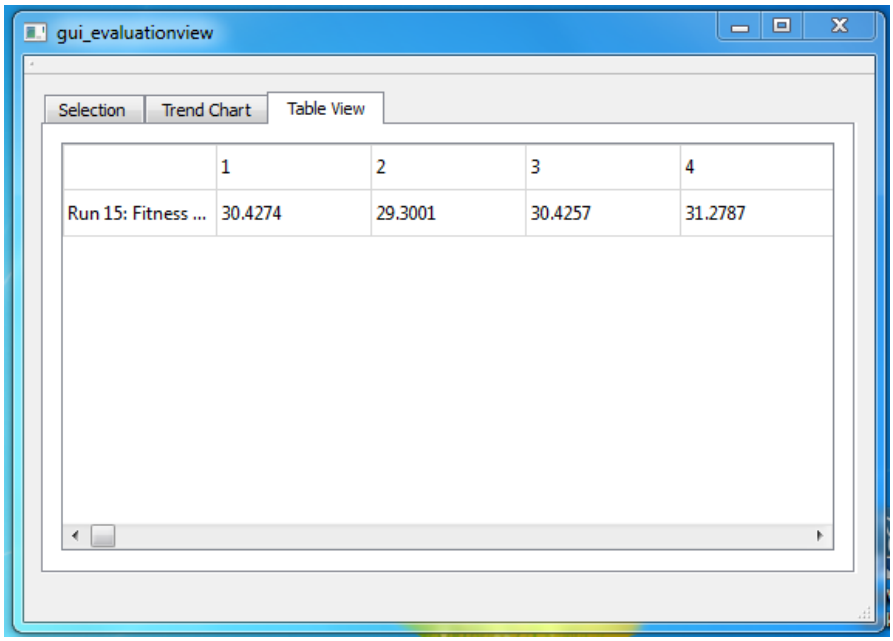


Abb. 6.19 Beispieldaten in der Tabellenansicht der Evaluation View

6.11.2.3 Plot

Das Plotten von Rundaten wird mit Hilfe des *qcustomplot*-Widgets von Emanuel Eichhammer realisiert. Hierbei ist auf der X-Achse des Graphen die Zeit abgebildet. Für Optimierungszyklen ist eine Dauer von 15 Minuten angenommen, sodass 24 Stunden mit 96 Optimierungszyklen gefüllt werden. Die Y-Achse enthält die abzubildenden Datenwerte. Beide Achsen werden anhand der vorhandenen Daten automatisch skaliert. Abbildung 6.20 zeigt beispielhaft eine durch die Klasse geplottete Funktion.

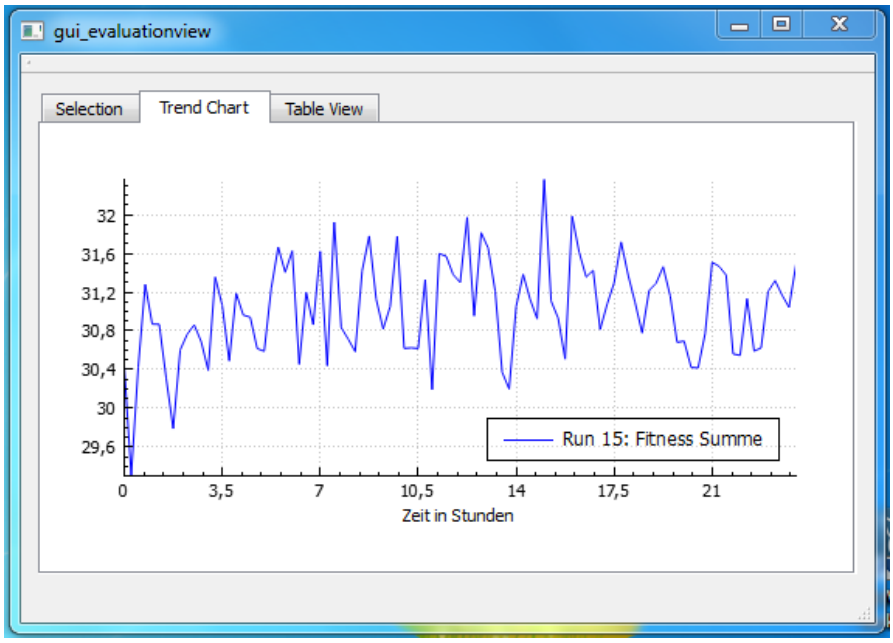


Abb. 6.20 Beispielplot der Evaluation View

Zu plottende Daten werden gesammelt als Paket an den Plot übergeben. Dieses Paket kann mehrere Funktionen enthalten, für welche jeweils ein Name, Datenwerte sowie ein Startzeitpunkt vorhanden sind. Das Paket wird mit einer Bezeichnung für die Y-Achse abgerundet, welche aufgrund der heterogenen Datentypen in unserer Implementierung ein leerer String ist.
Jedes so vorhandene Teilpaket wird beim Plotten in Einzelschritten abgearbeitet. Zunächst wird eine Funktion angelegt und mit ihrem Namen in der Legende bezeichnet. Anschließend werden die Datenwerte ab dem Startzeitpunkt als Funktionswerte eingetragen, sodass verschiedene Runs zeitlich unterscheidbar sind. Die frühesten Daten werden hierbei als Nullwert interpretiert. Abschließend wird eine Farbwahl

getroffen, sodass jede zu plottende Funktion eindeutig identifizierbar ist.
Die Farbauswahl orientiert sich an Permutationen des RGB-Raums, wobei die Intensität bei erreichen aller Kombinationen halbiert wird. Die Farbwerte pro Intensität berechnen sich als *(0, 0, X)*, *(0, X, 0)*, *(X, 0, 0)*, *(0, X, X)*, *(X, 0, X)* und *(X, X, 0)*. X stellt die Intensität der Farbe dar und beginnt somit bei dem Maximalwert 255.

Für einfachere Benutzung wurden einige Funktionen zur Handhabung aktiviert: Der Plot lässt sich beliebig in X- oder Y-Richtung verschieben und Zoomen. Weiterhin kann ausgewählt werden, dass die einzelnen Funktionswerte der ursprünglichen Datenreihen im Graph hervorgehoben werden können. Dies ist Standardmäßig allerdings deaktiviert. Zusätzlich wurde ein Eventhandler implementiert, welcher bei Auswahl einer Funktion oder eines Namens in der Legende das entsprechende Gegenstück ebenfalls markiert, sodass einzelne Funktionen einfach hervorgehoben werden können.
Für den Export der Graphen wird eine weitere Funktion der Bibliothek genutzt, welche eine PDF des aktuellen Graphen in seinen aktuellen Dimensionen im Unterordner *PlotData* erstellt. Die PDF ist mit einem Zeitstempel eindeutig identifiziert.

Kapitel 7
Softwareversionen

In diesem Abschnitt wird auf die wichtige Meilensteine der Entwicklung, der Weg vom ersten Prototypen zur finalen Version, eingegangen. Neben einer kleinen Übersicht über den Funktionsumfang werden auch spezielle Feinheiten der Implementierung erläutert. Dabei wird in Abschnitt 7.1 zunächst der erste Prototyp vorgestellt. Hierauf folgt mit Abschnitt 7.2 bereits die Alphaversion des Systems. Abschließend wird die zur Abgabe finale Version 1.0 in Abschnitt 7.3 beschrieben.

7.1 Prototyp

Wie im Kapitel 2.3 beschrieben wurde in der zweiten Projektphase ein Prototyp entwickelt. Da das Unified-Process-Vorgehensmodell ein iterativer Ansatz zur Softwareentwicklung ist, sieht es die Entwicklung eines Prototypen in den früheren Projektphasen vor. Das hat den Vorteil, dass Fehler bei der Systemarchitektur frühzeitig gefunden werden können.
In unserem Projekt haben wir uns entschieden einen Prototypen zu entwickeln, der ein einfaches Optimierungsbeispiel ausführen kann. Dieser Prototyp sollte dazu dienen

- die Programmierinfrastruktur anzulegen
- die Kommunikation zu implementieren
- jedem Projektmitglied die Möglichkeit zu bieten, sich in die Programmiersprache und das Entwicklungsumfeld einzuarbeiten.

Das Szenario des Prototyp war möglichst einfach gehalten, da kein zu großer Programmieraufwand während der Einarbeitungsphase aller Mitglieder entstehen sollte. Zunächst waren grundlegende Schnittstellen zur Kommunikation das wichtigste zu entwickelnde Element. Es wurde daher entschieden anstelle einer Optimierung zunächst die einfache Rechenproblematik des kleinsten gemeinsamen Vielfachen als

Platzhalter zu verwenden, um die implementierte Programmierstruktur und Kommunikation zu testen. Ein Zahlenbeispiel des Prototypen wird am Ende dieses Abschnitts beschrieben.

Die drei Hauptkomponenten, Energiemanagementsystem, Simulationsframework und Steuerungsebene, sollten dabei jeweils einen Teil der Berechnung übernehmen. Auch die Geräte von Beckhoff sollten im ersten Prototypen bereits zum Einsatz kommen. Damit sind alle Systeme eingebunden und erste Erfahrungen in allen Bereichen der Implementation konnten gemacht werden.

Die SE wurde in diesem Prototypen noch nicht als GUI sondern als Konsolenapplikation implementiert. Neben dem Ausführen eines Start- und Stoppsignals durch Senden an das SF und EMS, visualisiert sie die vom EMS empfangenen Daten. Zum Zeitpunkt des Prototyp war außerdem noch der Taktgeber in der Steuerungsebene enthalten und sendete die Taktsignale an EMS und SF.

Die vom EMS empfangenen Daten enthalten den Zustand der Komponenten, definiert durch eine Zufallszahl, einen Multiplikator und dessen Ergebnis, dem Optimierungsergebnis, definiert durch das kleinste gemeinsame Vielfache und den gewählten Multiplikator für die Komponenten, und die Auswertung, definiert durch eine einfache Multiplikation, die Gesamteinsparung.

Das Simulationsframework übernimmt im Prototyp die Aufgabe die simulierten EVS-Komponenten auf dem Beckhoffsystem zu initialisieren und im Ablauf des Szenarios mit jedem Taktsignal neue Zufallszahlen an die Komponenten zu schicken. Diese Zufallszahlen könnte man als das aktuelle Wetter betrachten, das von den Komponenten benötigt wird um ihren Zustand zu simulieren. Außerdem werden diese Zufallszahlen vom Simulationsframework in eine Datenbank geschrieben. Auch in Bezug auf die Datenbank gab es nach der Verwirklichung des ersten Prototypen Veränderungen in der Architektur. So wurde im Prototypen für das EMS und das SF jeweils auf die Datenbank (DB) zugegriffen und die Steuerungsebene erhielt benötigte Daten über das EMS bzw. SF.

Die Steuerungsebene schreibt dabei nur zu Beginn einmal das Szenario, Wetterprognosedaten und Grundlastdaten, falls noch nicht vorhanden, in die DB. Im weiteren Verlauf greift sie nur lesend auf die DB zu.

	Komponente 1	Komponente 2	Komponente 3
Zufallszahl	3	5	18

Tabelle 7.1 Zahlenbeispiel mit 3 EVS-Komponenten, Schritt 1: Das Simulationsframework erstellt die Zufallszahlen und sendet sie an die EVS-Komponenten

Die simulierten EVS-Komponenten führen in jedem Zeitabschnitt nur eine Berechnung durch. Durch das Ergebnis der Multiplikation von der Zufallszahl mit einer vorgegebenen Zahl, wird der Zustand der EVS-Komponente dargestellt. Der Zustand (Zufallszahl, Multiplikator, Multiplikationsergebnis) wird von den Komponenten an das SF gesendet. Diese speichert den Zustand der Komponente in seiner Datenbank ab. Auf Anfrage des EMS sendet die EVS-Komponente ihren Zustand.

	Komponente 1	Komponente 2	Komponente 3
Multiplikator	1	1	1
Zufallszahl	3	5	18
Ergebnis	3	5	18

Tabelle 7.2 Zahlenbeispiel mit 3 EVS-Komponenten, Schritt 2: Die EVS-Komponenten multiplizieren die Zufallszahl mit ihrem Multiplikator.

Das Energiemanagementsystem speichert den Zustand der EVS-Komponenten ebenfalls in seiner Datenbank. Das Optimierungssystem des EMS fordert die Zustände aller EVS-Komponenten zur Optimierung an. Optimierung bedeutet im Prototyp, dass sie zuerst für alle Multiplikatoren der Komponenten das kleinste gemeinsame Vielfache (kgV) berechnet. Und daraufhin für jede einzelne Komponente die Zahl sucht, die multipliziert mit ihrem Multiplikator das kgV ergibt. Diese Zahl wird als neuer Multiplikator an die EVS-Komponente gesendet und soll damit das Steuersignal darstellen. Der neue Multiplikator wird vom EMS in der DB gespeichert. Alle Zustände der EVS-Komponenten und Berechnungsergebnisse des Optimierungssystems werden an die Steuerungsebene gesendet.

	Komponente 1	Komponente 2	Komponente 3	kgV
Multiplikator	1	1	1	
Zufallszahl	3	5	18	90
Ergebnis	3	5	18	
opt. Multiplikator	30	18	5	

Tabelle 7.3 Zahlenbeispiel mit 3 EVS-Komponenten, Schritt 3: Die Optimierung des EMS berechnet das kleinste gemeinsame Vielfache. Die neuen Multiplikatoren der Komponenten ergeben sich aus der Division vom kgV durch die Zufallszahl.

Die Auswertungskomponente der Optimierung wertet das Ergebnis aus, indem sie die Differenz zwischen kgV der Zufallszahlen und einer einfachen Multiplikation der Zufallszahlen berechnet. Die Einsparung entspricht der Summe der Differenzen von dem gewählten Multiplikator zu dem bei der einfachen Multiplikation notwendigen Multiplikator.

	Komponente 1	Komponente 2	Komponente 3
Multiplikator	1	1	1
Zufallszahl	3	5	18
Ergebnis	3	5	18
opt. Multiplikator	30	18	5
unpot. Multiplikator	270/3 = 90	270/5 = 54	270 / 18 = 15
Differenz	90-30 = 60	54 - 18 = 36	15 - 5 = 10
kgV = 90, **Mult. der opt. Multiplikatoren** = 270, **Optimierungsdifferenzsumme = 106**			

Tabelle 7.4 Zahlenbeispiel mit 3 EVS-Komponenten, Schritt 4: Die einfache Multiplikation der Zufallszahlen ergibt 270. Die Summe der Differenzen zwischen unoptimierten und optimierten Multiplikatoren ergibt 106.

Die EVS-Komponenten erhalten die neuen Steuerungsbefehle vom EMS (optimierte Multiplikatoren). Tabelle 7.5 zeigt den neuen Zustand der Komponenten.

	Komponente 1	Komponente 2	Komponente 3
Zufallszahl	3	5	18
Multiplikator	30	18	5
Ergebnis	90	90	90

Tabelle 7.5 Zahlenbeispiel mit 3 EVS-Komponenten, Schritt 5: Die neuen Multiplikatoren wurden vom EMS an die EVS-Komponenten gesendet. Mit dem nächsten Taktsignal sendet das SF die neuen Zufallszahlen und der Vorgang beginnt von neuem.

7.2 Alpha-Version

Wie in Kapitel 2.3 erläutert, wurde statt eines zweiten Prototypen eine Alpha-Version entwickelt. Die Alpha-Version war am 11.12.2014 fertiggestellt. Teilbereiche des ersten Prototypen dienten bei der Entwicklung als Grundlage und wurden zur Alpha-Version weiterentwickelt. Dies war ohne großen Aufwand möglich, da beim ersten Prototypen schon sehr viel Planung in die Softwarearchitektur gesteckt wurde.

Im Unterschied zum ersten Prototypen funktionierten in der Alpha-Version schon wesentliche Teile des Gesamtsystems, sodass die Alpha-Version eine Grundlage für die Version 1.0 darstellt (vgl. Abschnitt 7.3). Für die Alpha-Version wurde die Optimierung von mindestens zwei EVS-Komponenten umgesetzt. Die Kommunikation der Subsysteme untereinander und zu den EVS-Komponenten wurde überarbeitet. Letztere wurde vollständig auf das etablierte Industrieprotokoll OPC-UA umgestellt. OPC-UA ermöglicht, dass die Kommunikation ohne großen Aufwand auch mit echten Hardwarekomponenten funktioniert. Des Weiteren wurde ein GUI implementiert,

die eine Steuerung des EMS und des SF ermöglicht und Komponentendaten sowie Informationen über ausgeführte Zyklen anzeigt. Somit wurde bereits ein Überblick über den Zustand des Gesamtsystems dargestellt.

Als EVS-Komponenten wurde bereits ein Energieerzeuger und ein steuerbaren Verbraucher realisiert: Für den Erzeuger fiel die Entscheidung auf ein Blockheizkraftwerk, während eine Waschmaschine als steuerbarer Verbraucher diente. Die Alpha-Version konnte eine rudimentäre Implementierung der Grundlast, die aus einer Datei ausgelesen und vom BHKW verwendet wurde, vorweisen. Die Berücksichtigung von Wetterdaten sollte in der Alpha-Version noch nicht beachtet werden und wurde dementsprechend nicht umgesetzt.

Bei der Datenbank wurden ebenfalls einige grundlegende Änderungen realisiert: Im Prototypen griffen lediglich das EMS und SF auf die Datenbank zu. In der Alpha-Version erhielt die Steuerungsebene ebenfalls direkten Zugriff auf die Datenbank: Die Steuerungsebene schreibt einmalig zu Beginn das Szenario sowie die Grundlastdaten, falls noch nicht vorhanden, in die Datenbank. Außerdem werden Konfigurationsparameter des Runs gespeichert. Im EMS und SF werden diese Daten ausgelesen und verwendet.

Der Taktgeber wurde ins EMS verlegt und angepasst. Dieser gibt dem EMS und dem SF den Takt zum Arbeiten vor. Über die Kommunikation wird sichergestellt, dass zunächst der Takt im SF und anschließend im EMS ausgeführt wird, bevor ein neuer Takt gestartet wird (siehe Abschnitt 5.12).

Während eine Beta-Version im Anschluss an die Alpha-Version geplant war, musste diese Planung angepasst werden (vgl. Abschnitt 2.3). Daher folgt auf die Alpha-Version direkt die Version 1.0. Dies lässt der Alpha-Version umso größere Bedeutung zukommen und bestätigt die Entscheidung, die wesentliche Funktionalität bereits in die Alpha-Version zu integrieren.

7.3 Version 1.0

Als erstes Release wurde, auf die Alpha-Version (siehe Abschnitt 7.2) aufbauend, die Version 1.0 entwickelt und schließlich am 29.01.2015 fertig gestellt.

In der Version 1.0 sind alle geforderten EVS-Komponenten enthalten. Als Neuentwicklung kamen dabei die PV-Anlage, die Batterie und ein großer steuerbarer Verbraucher hinzu. Für die PV-Anlage wurde der Projektgruppe ein bereits vorhandenes Modell zur Verfügung gestellt. Als steuerbarer Verbraucher ist ein Kühlhaus entwickelt worden, das in Größe und Energieverbrauch anpassbar ist.
Eine weitere Neuentwicklungen zum Vorgänger, der Alpha-Version, ist die Implementierung der Wetterdaten. In diesem Zusammenhang wurde auch die Grundlastberechnung überarbeitet. Statt der bisher rudimentären Weise, indem sie aus einer Datei gelesen wurde, ist die Grundlast anhand der Wetterdaten nach der Norm VDI 4655 (siehe Abschnitt 5.15) berechnet. Als Quelle für die Wetterdaten nutzen wir die Wetterstation der Universität Oldenburg. Da leider nicht alle benötigten Daten aus dieser Quelle gezogen werden können, musste eine zusätzliche Quelle für den

Bedeckungsgrad genutzt werden. Dafür nutzen wir die Daten einer vom DWD frei zugängliche Wetterstation in Bremen.
In der Optimierung wurde die Architektur angepasst, so dass der Algorithmus der Optimierung nun austauschbar ist. Die Austauschbarkeit des Algorithmus war eine der Anforderungen an das Projekt.
Die Kommunikation der Systeme wurde deutlich flexibler gestaltet, sodass z.B. zur Laufzeit für jedes System benötigte Sockets dynamisch erzeugt werden. Dies ermöglicht z.B. getrennte Steuersignale für das EMS und das SF.
Die Entwicklung einer Schnittstelle zur Evaluation des Gesamtsystems wurde in der Alpha-Version noch nicht beachtet. Daher wurde dieser Teilbereich erst in Version 1.0 entwickelt. Anpassungen dafür waren vor allem in der Datenbank notwendig. Die Laufdaten der Simulation und deren Komponenten sowie die Daten der Optimierung werden hierfür gespeichert. Neuentwicklung war ebenfalls die zur Evaluation genutzte GUI (siehe Abschnitt 6.11). Sie ermöglicht ein Offline-Evaluation zwischen verschiedene Läufen eines oder mehrerer Szenarien.
Zu Anpassungen und Erweiterungen kam es ebenfalls in der bereits verwendeten GUI der Alpha-Version. Die wichtigste Erweiterung ist das Anzeigen der Logging-Nachrichten in der GUI. Es werden die Log-Daten aller Systeme angezeigt, damit der Nutzer über den Ablauf dieser genau in Kenntnis gesetzt ist. Anpassungen wurden in den Wizards zur Erstellung von Szenarien und Läufen vorgenommen.
Auf Grund von zu wenig Planung und Absprache der vom jeweiligen System, Subsystem und Modul geforderten Einheiten, mussten nachträglich einige APIs oder deren Verwendung angepasst werden. Dieser Aufwand hätte durch bessere Kommunikation verhindert werden können.
Abschließend kam es nach der Integration aller Neuentwicklungen zum Testen der Version 1.0. Damit einhergehend ist das Bugfixing aller auftretenden Fehler. Ein besonderes Interesse galt der Datenbank und ihrer funktionalen Anwendung. Entstandene Performanzprobleme durch Schreiben oder Lesen von nicht notwendigen Daten führten zu erheblichen Zeitverzögerungen, bis hin zur Unbenutzbarkeit der Evaluations-GUI. Diese Problematik wurde während der Testphase behoben, so dass eine Vernünftige Verwendung aller Systeme gegeben ist.

Kapitel 8
Anforderungserfüllung

Um das entwickelte System bewerten zu können, muss es mit den festgelegten Anforderungen aus Abschnitt 4.3 verglichen werden. Ziel ist es dabei, eine Aussage darüber zu machen, in wie weit das gewünschte System dem entwickelten System entspricht. Um dies zu erreichen, wird für jede einzelne Anforderung festgestellt, in welchem Maß diese im Endprodukt umgesetzt wurde.

Dafür wird jede Anforderung noch einmal mit Namen und Beschreibung aufgelistet und einzeln untersucht. Es wird die Art der Darstellung aus Abschnitt 4.3 übernommen und um den Eintrag Umsetzung ergänzt. Bei der Umsetzung gibt es drei unterschiedliche Kategorisierungen und zwar vollständig, teilweise und nicht erfüllt. Jeder Anforderung wird eine dieser Kategorien zugeordnet. Dabei soll kurz begründet werden, weshalb die Anforderung dieser Kategorie entspricht. Bei teilweise und nicht erfüllten Anforderungen soll dabei explizit auf die Schwierigkeiten bei der Realisierung der Anforderung eingegangen werden.

8.1 Energiemanagementsystem

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario (EMS) konfigurieren	EMS-10A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Möglichkeit bieten das Szenario für das Energiemanagementsystem zu konfigurieren. Außerdem muss eine Datenquelle für eine Grundlastprognose angegeben werden.		
Umsetzung	vollständig		

Dies war eine der Grundanforderungen des Projektvorhabens und essentiell für die Umsetzung einer Optimierung im Energiemanagementsystem. Als Datenquelle

für die Grundlastprognose dient die Datenbankanbindung. Weitere Konfigurationsparameter sind der Preis für den Ankauf bzw. Verkauf für elektrische Energie, die Art und die Anzahl der EVS-Komponenten sowie die Cycletime, mit der der Taktgeber initialisiert wird. Diese Parameter werden vom Anwender in der Steuerungsebene festgelegt und von dieser in der Datenbank abgespeichert. Bei der Konfiguration des EMS werden diese Parameter aus der Datenbank ausgelesen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsziel festlegen	EMS-15A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Möglichkeit bieten, das Optimierungsziel festzulegen. Das Optimierungsziel bestimmt auf welche Art die Optimierung aus den Samples der EVS-Komponenten einen Fahrplan auswählt.		
Umsetzung	vollständig		

Das Optimierungsziel ist im Energiemanagementsystem festlegbar. Allerdings kann nur ein Optimierungsziel ausgewählt werden. Dieses lautet *Eigenverbrauchsoptimierung zur Minimierung der Energiebezugskosten*. An dieser Stelle muss betont werden, dass damit nicht einher geht, dass das Optimierungsziel über die grafische Oberfläche der Steuerungsebene auswählbar ist. Dafür gibt es die Anforderung SE-EMS-25. Weitere Optimierungsziele können auf einfache Weise hinzugefügt werden, indem eine neue Klasse definiert wird. Dafür bietet die Optimierung die Schnittstelle *IFitnessCalculator*, die die Aufgabe hat, zu einem Array von Samplings die Fitness zu berechnen (siehe Abschnitt 6.9).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlast (thermisch, elektrisch) abrufen	EMS-30A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss Informationen über die Grundlast (thermisch, elektrisch) des Gebäudes abrufen können. Eine Schnittstelle für das Abrufen der Daten muss im EMS enthalten sein. Im Falle der Simulation gibt es eine Quelle für die Grundlast, die mit der Konfiguration des Szenarios integriert wird und über das Simulationsframework verfügbar ist.		
Umsetzung	vollständig		

Die Grundlast steht dem Energiemanagementsystem und damit der Optimierung über die Datenbankanbindung zur Verfügung. Es wird nicht zwischen dem normalen Betrieb und dem Betrieb in einer Simulation unterschieden. Dadurch entfällt im Falle einer Simulation eine Konfiguration des Energiemanagementsystems mit der Quelle

für die Grundlast. Für eine Simulation wird die Grundlast in der Steuerungsebene wie in Abschnitt 5.15 beschrieben, berechnet.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsschritt	EMS-60A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss für die steuerbaren EVS-Komponenten anhand der Samples und gemäß dem gewählten Optimierungsziel den folgenden Betriebsschritt aller EVS-Komponenten bestimmen.		
Umsetzung	vollständig		

Der folgende Betriebsschritt aller EVS-Komponenten wird im Energiemanagementsystem durch die Optimierung unter Berücksichtigung des Optimierungsziels ausgewählt. Dabei bestimmen die EVS-Komponenten durch das Sampling eine Menge von möglichen Betriebsschritten (vgl. Abschnitt 6.7), die der Optimierung als Eingabe dienen. Wird das Sampling korrekt ausgeführt, so verletzt ein Betriebsschritt keine Gerätebedingungen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Komponentenfahrplan anpassen	EMS-70	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss den Fahrplan für die steuerbaren EVS-Komponenten im vom Systemtakt vorgegebenen Abstand aktualisieren, d.h. neu optimieren.		
Umsetzung	vollständig		

Im fertigen System findet in jedem Systemtakt ein neues Sampling der EVS-Komponenten statt. Das Ergebnis des Samplings jeder EVS-Komponente wird dem Energiemanagementsystem zur Verfügung gestellt. Die Optimierung im Energiemanagementsystem wählt daraufhin einen Fahrplan für alle EVS-Komponenten aus, wobei damit auch der nächste Betriebsschritt der Komponenten bestimmt ist. Da in jedem Systemtakt ein neues Sampling der EVS-Komponenten stattfindet, hat der Fahrplan oft nur eine Gültigkeit von einem Systemtakt. Eine Ausnahme gibt es nur dann, wenn der Fahrplan durch das Sampling nochmal generiert und durch die Optimierung erneut gewählt wird.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Steuerungssignale versenden	EMS-80A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Wahl eines Fahrplans an die EVS-Komponenten übermitteln.		
Umsetzung	vollständig		

Hat die Optimierung im Energiemanagementsystem einen Fahrplan ausgewählt, so wird dieser vom EMS an die entsprechende EVS-Komponente weitergeleitet. Dafür wurde eine Kommunikation mit den Komponenten über die Schnittstelle OPC-UA ermöglicht. (vgl. Abschnitt 6.2)

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Ergebnis der Optimierung bereitstellen	EMS-90A	geschlossen	unkritisch
Beschreibung	Das Energiemanagementsystem muss das Ergebnis der Optimierung bereitstellen. Das Ergebnis der Optimierung kann dem Anwender mittels Visualisierungskomponente dargestellt werden und setzt sich als eine Kostenprognose für die nächsten 24 Stunden zusammen.		
Umsetzung	vollständig		

Das Ergebnis der Optimierung wird in der Datenbank nach jedem Systemtakt abgespeichert. Dazu wird zum einen die Summe der Kostenprognose für die nächsten 24 Stunden und zum anderen die 96 Einzelwerte, die die Kosten für jeweils 15 Minuten angeben, abgespeichert. Der Anwender kann dies zur Offline-Evaluation nutzen. Die Steuerungsebene kann mit einem Zugriff auf die Datenbank diese Informationen abrufen und sie visualisieren. Die entsprechende Anforderung dazu lautet *Statistik der Kostenprognose visualisieren* (siehe SE-EMS-15).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Samplings der EVS-Komponenten abrufen	EMS-100A	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss die Samplings der EVS-Komponenten auslesen oder empfangen können.		
Umsetzung	vollständig		

Damit die Optimierung die Fahrpläne bzw. die folgenden Betriebsschritte der EVS-Komponenten bestimmen kann, muss sie die Samplings zur Verfügung haben. Die Erfüllung dieser Anforderungen ist damit eine notwendige Voraussetzung für die Erfüllung der Anforderungen EMS-60A sowie EMS-70. Die Samplings der EVS-

Komponenten werden wie auch die Steuerungssignale über die Schnittstelle OPC-UA übermittelt.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsschritt bei Simulation vorzeitig beenden	EMS-120A	geschlossen	optional
Beschreibung	Das Energiemanagementsystem soll die Möglichkeit bieten, während einer Simulation einen Optimierungsschritt vorzeitig zu beenden.		
Umsetzung	vollständig		

Die Funktionalität den laufenden Optimierungsschritt vorzeitig zu beenden, ist im Energiemanagementsystem implementiert. Allerdings gibt es zu einem Zeitpunkt keine Information darüber, wie gut die Fitness der aktuell besten Lösung ist. Damit ist es dem Anwender nicht möglich, einen sinnvollen Zeitpunkt zum vorzeitigen Beenden auszuwählen. Außerdem muss der Taktgeber darüber informiert werden, dass der nächste Systemtakt vorgezogen wird, was nicht geschieht. Zudem ist diese Funktionalität nicht in der Steuerungsebene verfügbar. Damit ist zwar das Akzeptanzkriterium der Anforderung erfüllt (vgl. EMS-120A), allerdings ohne weitere Funktionalität für das Gesamtsystem zu schaffen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Samplings speichern	EMS-140A	geschlossen	unkritisch
Beschreibung	Das Energiemanagementsystem muss die gewählten Samplings zu jedem Zeitschritt speichern können.		
Umsetzung	teilweise		

Aufgrund der Datenmenge hat sich die Projektgruppe dazu entschieden, nicht die von der Optimierung gewählten Samplings abzuspeichern. Stattdessen speichert das Energiemanagementsystem nach jedem Zeitschritt die Indizes der gewählten Samplings im Log des EMS ab. Dies war für Debuggingzwecke ausreichend, auch wenn es nach einem Simulationsschritt keinen Rückschluss auf die tatsächlichen Betriebseinstellungen der EVS-Komponenten mehr erlaubt.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Interne Log erstellen	EMS-150	geschlossen	unkritisch
Beschreibung	Das Energiemanagementsystem muss eine Logdatei erstellen. Diese muss Informationen über Daten, Funktionsaufrufe und Fehler enthalten.		
Umsetzung	vollständig		

Das Energiemanagementsystem legt eine Logdatei an. Diese Logdatei wird auf dem Rechner gespeichert, der das EMS ausführt. Zudem wird das Log über die Kommunikation direkt an die Steuerungsebene versendet und dort angezeigt. Dabei werden jeweils nur die neuen Zeilen des Logs versendet und nicht die komplette Datei.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Kommunikationslog erstellen	EMS-160	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss ein Kommunikationslog erstellen. Dieses dokumentiert die ein- und ausgehende Kommunikationen.		
Umsetzung	vollständig		

Das Energiemanagementsystem legt ein Kommunikationslog an. Das Kommunikationslog wird dabei in die selbe Datei geschrieben, wie das interne Log. Analog zum internen Log wird auch das Kommunikationslog über die Kommunikation an die Steuerungsebene versendet.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Logdatei des Energiemanagementsystems bereitstellen	EMS-180	geschlossen	kritisch
Beschreibung	Das Energiemanagementsystem muss Logdateien bereitstellen. Die Logdateien informieren über den Zustand und die Historie des Energiemanagementsystems.		
Umsetzung	vollständig		

Bei der Initialisierung des Energiemanagementsystem wird eine Logdatei auf dem ausführenden Rechner angelegt. Diese Logdatei speichert sowohl das Kommunikationslog als auch das interne Log.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EMS beenden	EMS-170	geschlossen	kritisch
Beschreibung	Das EMS muss vom Anwender über die Steuerungsebene beendet werden können. Die EVS-Komponenten erhalten vom Energiemanagementsystem keine weiteren Steuersignale.		
Umsetzung	vollständig		

Das EMS kann vom Anwender über die Steuerungsebene beendet werden. Dafür wird ein Stopp-Signal von der Steuerungsebene versendet. Das EMS stellt darauf seinen Betrieb ein. Es bleibt aber ansprechbar und kann mit einem anderen Szenario initialisiert werden. Empfängt das EMS ein Stopp-Signal, so stellen die EVS-Komponenten ihren Betrieb ein, da sie die notwendigen Steuersignale vom EMS nicht mehr erhalten.

8.2 Simulationsframework

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Verrauschen der simulierten Wetterprognosedaten	SF-40A	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, die Wetterprognosedaten zu verrauschen. Damit soll sichergestellt werden, dass das simulierte Wetter von der Wetterprognose abweicht.		
Umsetzung	vollständig		

Die Wetterprognosedaten werden im Simulationsframework verrauscht. Anschließend werden die verrauschten Wetterprognosedaten, das simulierte Wetter, in der Datenbank abgespeichert. Dies passiert für alle Tage bei der Initialisierung des Simulationsframeworks mit Hilfe einer Zufallsfunktion. Um die Möglichkeit zu haben, die generierten Werte der Zufallsfunktion erneut zu erzeugen, wird ein *Seed* als Startwert übergeben. Die *Seed* wird in der Steuerungsebene vom Anwender bei der Erstellung des Szenarios gewählt.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Verrauschen der Grundlast	SF-55	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, die Grundlast zu verrauschen. Analog zum Verrauschen der Wetterprognosedaten, wird damit eine Abweichung der tatsächlichen thermischen und elektrischen Last des Gebäudes von der prognostizierten Grundlast ermöglicht.		
Umsetzung	vollständig		

Die thermische und elektrische Grundlast wird direkt bei Initialisierung des Simulationsframeworks für alle Zeitschritte verrauscht und in der Datenbank gespeichert. Dies geschieht auf die selbe Weise, wie die Verrauschung der Wetterprognosedaten. Der verwendete Seed für die Zufallsfunktion ist der gleiche.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario initialisieren (SF)	SF-60	geschlossen	kritisch
Beschreibung	Das Simulationsframework initialisiert das Szenario.		
Umsetzung	vollständig		

Nach der Konfiguration des Szenarios durch den Anwender in der Steuerungsebene ist dieses in der Datenbank abgespeichert. Wird das Simulationsframework gestartet, lädt es das Szenario aus der Datenbank. Anschließend initialisiert es die EVS-Komponenten mit den übergebenen Parametern des Anwenders. Danach ist die Simulation startbereit.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation implizit starten	SF-70A	geschlossen	kritisch
Beschreibung	Das Simulationsframework startet bei *mit Simulation* konfiguriertem EMS ebenfalls die Simulation.		
Umsetzung	vollständig		

Das Simulationsframework startet die Simulation nach Eingang des ersten Taktsignals über die Kommunikation. Jeder weitere Simulationsschritt erfolgt nach einem Taktschritt. Die Aufgabe des Simulationsframesworks für jeden Simulationsschritt besteht darin, die EVS-Komponenten mit den notwendigen Eingaben der Simulation zu versorgen. Die notwendigen Eingaben einer EVS-Komponente sind in der Datenbank abgespeichert.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation separat starten	SF-80A	geschlossen	optional
Beschreibung	Das Simulationsframework bietet die Möglichkeit, die Simulation ohne das EMS zu starten.		
Umsetzung	nicht erfüllt		

Es ist nicht möglich, einen nicht optimierten Verlauf zu simulieren. Die EVS-Komponenten bieten kein angemessenes Standardverhalten. Es muss wenigstens bestimmt werden, welche EVS-Komponente zu einem Zeitpunkt die aktuelle Energienachfrage erfüllen soll bzw. was geschehen soll, wenn zu wenig oder zu viel Energie erzeugt wird. Im implementierten System ist es sogar notwendig, dass das EMS für die simulierten EVS-Komponenten ein Flag zum Starten setzen muss. Dieses Flag wird von den EVS-Komponenten nach jedem Betriebsschritt zurückgesetzt. Das EMS signalisiert mit diesem Flag, dass die Optimierung den folgenden Betriebsschritt ausgewählt hat und dieser auch zur EVS-Komponente übertragen wurde. Zudem benötigt das Simulationsframework das Taktsignal des Taktgebers im Energiemanagementsystem, um einen Simulationsschritt auszuführen. Der Taktgeber im Energiemanagementsystem ist allerdings austauschbar gehalten und kann leicht gegen einen eigenständigen Taktgeber ausgetauscht werden.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EVS-Komponenten simulieren	SF-90	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die für das Szenario konfigurierten EVS-Komponenten simulieren.		
Umsetzung	vollständig		

Das Simulationsframework versorgt die EVS-Komponenten vor einem Betriebsschritt über die Kommunikationsschnittstelle mit den jeweils notwendigen Eingabeparametern und signalisiert dies über ein entsprechendes Flag. Sie führen den nächsten Betriebsschritt aus, wenn auch das zweite notwendige Flag des Energiemanagementsystem gesetzt wurde. Alle notwendigen Parameter einer EVS-Komponenten für die Initialisierung und für einen Betriebsschritt sind in der Datenbank gespeichert.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Taktsignal empfangen	SF-110	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss ein Taktsignal empfangen können und führt auf Grund des Signals den folgenden Simulationsschritt durch.		
Umsetzung	vollständig		

Im Energiemanagementsystem und im Simulationsframework gibt es jeweils ein Teilsystem namens *ClockCommunication*. Im EMS dient dieses dazu, das Taktsignal des Taktgenerators zu versenden. Die *ClockCommunication* im SF empfängt dieses Signal und führt aufgrund dessen den nächsten Simulationsschritt aus.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Wetterprognosedaten und Wetter versenden	SF-130A	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, das simulierte Wetter sowie die Wetterprognose versenden zu können.		
Umsetzung	vollständig		

Das Simulationsframework kann das simulierte Wetter bzw. die Wetterprognose an die entsprechenden EVS-Komponenten versenden. Dies ist notwendig, um die EVS-Komponenten simulieren zu können (vgl. SF-90-Erfüllung).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlast und simulierte Wetter speichern	SF-135	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit bieten, den simulierten thermischen und elektrischen Energiebedarf des Gebäudes sowie das simulierte Wetter persistent zu speichern.		
Umsetzung	vollständig		

Sowohl der simulierte thermische und elektrische Energiebedarf des Gebäudes als auch das simulierte Wetter werden bei der Initialisierung des Simulationsframework verrauscht (vgl. SF-40A-Erfüllung und SF-55-Erfüllung) und anschließend automatisch in der Datenbank abgespeichert.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Kommunikationslog erstellen	SF-140	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss für die gesamte Kommunikation (vor und während der Simulation) ein Log erstellen.		
Umsetzung	vollständig		

Das Simulationsframework legt eine Logdatei an. In dieser Datei wird die Kommunikation erfasst. Analog zum Logging im Energiemanagementsystem wird in diese Datei auch das interne Log geschrieben. Zudem ist das Log über die Kommunikation in der Steuerungsebene verfügbar.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Internes Log erstellen	SF-150	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss ein systeminternes Log erstellen.		
Umsetzung	vollständig		

Diese Anforderung ist vollständig erfüllt. Siehe dazu Anforderung SF-140-Erfüllung.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation implizit beenden	SF-160	geschlossen	optional
Beschreibung	Das System beendet die Simulation implizit, wenn das EMS beendet wurde.		
Umsetzung	vollständig		

Wird das Energiemanagementsystem beendet, so wird auch der dort enthaltene Taktgenerator beendet, sodass kein neues Taktsignal mehr generiert und versendet werden kann. Da das Simulationsframework damit kein Taktsignal mehr empfängt, stellt es die Simulation ein. Wird das EMS neu gestartet, so ist allerdings auch ein Neustart des SF notwendig.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulation separat beenden	SF-170	geschlossen	optional
Beschreibung	Bei separatem Start der Simulation muss diese auch separat beendet werden können.		
Umsetzung	nicht erfüllt		

Da die Simulation nicht separat gestartet werden kann (vgl. SF-80A-Erfüllung), ist es auch nicht notwendig, die Simulation separat zu beenden.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Logdatei des Simulationsframework bereitstellen	SF-180	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss Logdateien bereitstellen. Die Logdateien informieren über den Zustand und die Historie der Simulation und der in dem Szenario enthaltenen EVS-Komponenten.		
Umsetzung	vollständig		

Diese Anforderung ist vollständig erfüllt. Siehe dazu Anforderung SF-140-Erfüllung.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Simulationsdaten exportieren	SF-200	geschlossen	kritisch
Beschreibung	Das Simulationsframework muss die Möglichkeit geben, die Simulationsdaten zu exportieren. Die Simulationsdaten beinhalten die Simulationsergebnisse (Historie aller EVS-Komponenten und deren Zustände während der Simulation) und das Kommunikationslog der EVS-Komponenten.		
Umsetzung	teilweise		

Die Simulationsergebnisse der EVS-Komponenten setzen sich aus den genierten Samplings sowie den Ausgaben für die thermische und elektrische Last zusammen. Die Zustände der EVS-Komponenten sind streng genommen durch die Menge der internen Variablen gegeben. Das Simulationsframework greift weder auf die Samplings noch auf die Ausgaben zu. Diese Daten sind im Energiemanagementsystem verfügbar. Es werden aber aufgrund der Datenmenge lediglich die Ausgaben für die thermische und elektrische Last in der Datenbank abgespeichert (vgl. EMS-140A-Erfüllung). Die Zustände der EVS-Komponenten werden nicht ausgelesen. Bei der Kommunikation zwischen den EVS-Komponenten und dem EMS werden Fehler mit Hilfe des Loggers festgehalten.

8.3 Steuerungsebene

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EMS starten	SE-EMS-05	geschlossen	kritisch
Beschreibung	Das EMS muss vom Anwender über die Steuerungsebene gestartet werden können. Das Energiemanagementsystem wird mit den Daten des Szenarios initialisiert. Nach der Initialisierung kann die Optimierung starten.		
Umsetzung	vollständig		

Um das Energiemanagementsystem zu starten, wird ein Init-Signal von der Steuerungsebene über die Kommunikation versendet. Dabei wird auch die ID des Szenarios übermittelt. Das EMS ließt darauf hin das in der Datenbank gespeicherte Szenario aus und wird damit konfiguriert (vgl. EMS-10A-Erfüllung). Nach einem Start-Signal beginnt der Taktgenerator mit der Generierung von Taktsignalen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsstatus visualisieren	SE-EMS-10A	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss den Optimierungsstatus des Energiemanagementsystems abrufen und visualisieren können.		
Umsetzung	vollständig		

Als Optimierungsstatus bezeichnet die Projektgruppe die Information darüber, ob ein Cycle beendet wurde. Diese Information wird in der GUI ausgeben. Zusätzlich dazu wird vermerkt, wie viele Generationen die Optimierung zum Finden der Lösung gebraucht hat. Eine prozentuale Anzeige über den Berechnungsfortschritt der Optimierung in einem Cycle wurde nicht umgesetzt, da die Zeit, die ein Cycle in Anspruch nimmt, in der Regel der sehr kurz ist. Ist die Simulation und damit die Optimierung beendet, erfolgt ebenfalls eine Ausgabe.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Statistik der Kostenprognose visualisieren	SE-EMS-15	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss das Ergebnis der Optimierung des Energiemanagementsystems abrufen und visualisieren können.		
Umsetzung	vollständig		

Es wird nach jedem Taktschritt die aktuelle Kostenprognose im Logfenster ausgegeben. Zudem ist es möglich, die Kostenprognose mehrere Taktschritte grafisch anzuzeigen und zu vergleichen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
EVS-Komponentenzustände visualisieren	SE-EMS-20	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender Einsicht in den Status der EVS-Komponenten bieten. Der Zustand aller Verbraucher, Erzeuger und Speicher muss dargestellt werden. Die aktuellen Werte müssen den Anlagen spezifisch angezeigt werden (z.B. tabellarisch). Außerdem soll eine Historie über Planungs- und Störereignisse angezeigt werden.		
Umsetzung	vollständig		

Nach jedem Taktschritt wir der aktuelle Energiebedarf bzw. die aktuelle Energieproduktion (thermisch/elektrisch) jeder EVS-Komponenten angezeigt. Sollten Störereignisse auftreten, wie beispielsweise ein Ausfall einer EVS-Komponenten, wird eine Fehlermeldung ausgegeben. In diesem Fall stoppt das System.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Optimierungsziel auswählen	SE-EMS-25	geschlossen	optional
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit bieten, das Optimierungsziel auszuwählen. Anschließend muss dem Energiemanagementsystem das gewählte Optimierungsziel mitgeteilt werden.		
Umsetzung	nicht erfüllt		

Die Optimierung unterstützt nur ein Optimierungsziel (vgl. EMS-15A-Erfüllung). Die Projektgruppe hat sich deshalb gegen die Auswahlmöglichkeit in der Steuerungsebene entschieden, da dadurch keine neue Funktionalität entstanden wäre.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Zeitraffer einstellen	SE-10	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit bieten, den Zeitraffer für die Ablaufgeschwindigkeit der Simulation und Optimierung einzustellen.		
Umsetzung	vollständig		

Der Zeitraffer für die Ablaufgeschwindigkeit (Cycletime) wird in der Steuerungsebene bei der Definition eines Szenarios eingegeben. Dies ist ein ganzzahliger Wert, der die Dauer in Sekunden angibt. Analog zu den anderen Eingabeparameter eines Szenarios wird er vor dem Start einer Simulation in der Datenbank abgespeichert, sodass das Energiemanagementsystem die Optimierung mit diesem Parameter initia-

lisieren kann. Ein geeigneter Wert für die Ablaufgeschwindigkeit ist 15 Sekunden (vgl. Abschnitt 9.6.1).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Gerätepool anlegen	SE-50	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, einen Gerätepool anzulegen. Dazu gehört das Hinzufügen, Ändern und Löschen von Geräten. Unter dem Begriff Geräte werden alle EVS-Komponenten verstanden. Die integrierten Geräte müssen mit Defaultwerten definiert und persistent gespeichert werden.		
Umsetzung	vollständig		

In der Steuerungsebene können für die Definition eines Szenarios EVS-Komponenten aus den zur Verfügung stehenden Typen ausgewählt werden. Die gewählten EVS-Komponenten können mit spezifischen Parametern versehen werden. Anschließend werden sie mit dem vom Anwender definierten Szenario in der Datenbank persistent abgespeichert. Es ist allerdings nicht möglich einem Szenario mehrere EVS-Komponenten vom selben Typ hinzuzufügen. Dafür müssen die entsprechenden Projektdateien angepasst werden, die das Sampling und die Betriebsschritte der Komponenten auf den Kopplern festlegen.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario (Simulation) konfigurieren	SE-60	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, ein Szenario für die Simulation zu konfigurieren. Die aus dem Gerätepool ausgewählten EVS-Komponenten müssen mit spezifischen Parameter und Initialwerten gefüllt werden. Zu den EVS-Komponenten gehören Energieerzeuger, -verbraucher und -speicher. Es muss eine Dauer für das Szenario eingegeben werden. Außerdem muss die Geschwindigkeit für den Zeitraffer angegeben werden.		
Umsetzung	teilweise		

In der Steuerungsebene kann ein Szenario konfiguriert werden. Dabei werden EVS-Komponenten ausgewählt und mit spezifischen Parametern versehen. Die Dauer eines Szenarios hängt von der Länge der Grundlastdaten bzw. der Länge der Wetterprognosedaten und kann nicht explizit eingegeben werden. Die Geschwindigkeit für den Zeitraffer wird übergeben (siehe SE-10 -Erfüllung) und weitere notwendige Parameter für die Konfiguration eines Szenarios wie z.B. der Preis für den Ankauf bzw. Verkauf für elektrische Energie.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Wetterprognosedaten dem Szenario (Simulation) hinzufügen	SE-70	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, einem konfigurierten Szenario Wetterprognosedaten hinzufügen zu können.		
Umsetzung	vollständig		

Einem Szenario können bei seiner Konfiguration Wetterprognosedaten hinzugefügt werden. Dabei stehen verschiedene Datensätze zur Verfügung, die in der Steuerungsebene ausgewählt werden können. Die Wetterprognosedaten geben dabei das Wetter in einem viertelstündigen Takt an und beziehen sich mindestens auf einen ganzen Tag. Sie unterscheiden sich im Datum und der Uhrzeit, der Temperatur, dem Bewölkungsgrad sowie der Globalstrahlung des Tages.

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Grundlastmodell (Wärmebedarf, Strombedarf) dem Szenario (Simulation) hinzufügen	SE-80	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss dem Anwender die Möglichkeit geben, dem Szenario ein Grundlastmodell hinzufügen zu können.		
Umsetzung	teilweise		

Einem Szenario kann nicht explizit ein Grundlastmodell hinzugefügt werden. Die Grundlast wird in der Steuerungsebene anhand der Eingaben des Anwenders berechnet. Die notwendigen Eingaben sind die Wetterprognosedaten, der Jahresbedarf an thermischer und elektrischer Energie sowie der Simulationszeitraum. Als Ausgabe liegt der Energiebedarf des Gebäudes zu einem Tag vor. Das Gebäude stellt dabei ein Mehrfamilienhaus dar, dessen Größe durch die Parameter für den Jahresbedarf an elektrischer und thermischer Energie sowie einen Parameter für die Anzahl der Wohneinheiten gegeben ist. Diese Parameter werden über die Steuerungsebene eingegeben. Die Berechnung für die Grundlast erfolgt dabei nach der Norm VDI4655 (siehe Abschnitt 5.15).

Titel	*Anforderungsnr.*	*Status*	*Priorität*
Szenario persistent speichern	SE-90	geschlossen	kritisch
Beschreibung	Die Steuerungsebene muss ein Szenario persistent speichern können. Die Szenariodaten enthalten die ausgewählten EVS-Komponenten und deren Parameter. Bei Simulationsszenarien kommen die ausgewählten Wetterprognosedaten, Geschwindigkeit des Zeitraffers und das Grundlastmodell hinzu.		
Umsetzung	vollständig		

Das Szenario wird vom Anwender über die Steuerungsebene definiert. Damit auch das Simulationsframework und das Energiemanagementsystem auf die gewünschten Parameter Zugriff haben, wird das Szenario in der Datenbank mit einer ID persistent abgespeichert. Es kann anschließend über die Steuerungsebene erneut geladen werden.

8.4 Auswertung

Von allen 40 erfassten Anforderungen sind 33 vollständig erfüllt worden, was einem Anteil von 82,5 % entspricht. Lediglich drei Anforderungen wurden im entwickelten System nicht beachtet. Diese drei Anforderungen waren allerdings nicht für ein lauffähiges System notwendig, sondern als optional gekennzeichnet. Zwei dieser Anforderungen bezogen sich auf die Simulation eines ungesteuerten, nicht optimierten Verbrauchs (vgl. SF-80A-Erfüllung, SF-170-Erfüllung). Eine Simulation die also ohne das EMS lauffähig ist.

Diese Anforderungen waren unzureichend beschrieben, denn in irgendeiner Weise muss für jede EVS-Komponente der folgende Betriebsschritt bestimmt werden. Nicht alle EVS-Komponenten können nur mit den Eingabeparametern der thermischen und elektrischen Last vollständig gesteuert werden. Außerdem muss festgelegt werden, welche Komponenten bevorzugt Energie erzeugen oder verbrauchen dürfen. Damit die Energienachfrage nicht von allen Komponenten gleichzeitig gedeckt wird, muss eine Aufteilung dieser auf die Komponenten stattfinden. Dies allerdings stellt bereits eine Steuerung dar (Verteilung der Last), die nach dem Prinzip der Aufteilung in die drei Subsysteme vom EMS vorgenommen werden müsste. Es ist also sinnvoll, die Anforderungen neu zu verfassen und dabei die geforderte Funktionalität gegenüber der nicht gebrauchten Funktionalität abzugrenzen. Im konkreten Fall muss also definiert sein, was unter einem ungesteuertem, nicht optimierten Verbrauch einer EVS-Komponente verstanden wird.

Die dritte nicht erfüllte Anforderung (SE-EMS-25-Erfüllung), bezieht sich auf die Auswahlmöglichkeit eines Optimierungsziels in der Steuerungsebene. Da der Optimierung nur ein Optimierungsziel zur Verfügung steht (vgl. EMS-15A-Erfüllung),

bringt eine Auswahlmöglichkeit in der Steuerungsebene keine Mehrfunktionalität. Eine Umsetzung der Anforderungen war also aus zeitlichen Gründen nicht sinnvoll.

Obwohl vier Anforderungen nur teilweise erfüllt wurden, sind dadurch nur zwei wesentliche Funktionen verloren gegangen. In SE-60-Erfüllung wird verlangt, dass die Dauer eines Szenarios in der Steuerungsebene eingegeben werden kann. Diese hängt im entwickelten System allerdings von der Länge der Grundlastdaten bzw. der Länge der Wetterprognosedaten ab. Die in der Anforderung SE-80-Erfüllung verlangte Funktionalität, einem Szenario ein Grundlastmodell hinzufügen zu können, wurde durch eine erweiterte Sicht auf die Simulation nicht umgesetzt. Um die Grundlast auch vom simulierten Wetter abhängig zu halten, wird sie im entwickelten System anhand dieser Eingabe und dem Jahresbedarf an Energie des zu simulierenden Gebäudes berechnet. Diese Umsetzung sorgt dafür, dass die Grundlast im Verhältnis zum Wetter steht. Sie ist damit realistischer.

Die anderen beiden teilweise erfüllten Anforderungen beziehen sich auf das Abspeichern von Samplings EMS-140A-Erfüllung und das Exportieren von Simulationsdaten SF-200-Erfüllung. In beiden Fällen war die gewünschte Funktionalität weder für Debuging-Zwecke noch für Simulationszwecke notwendig. Die Projektgruppe hat sich aus Zeitgründen gegen die Umsetzung entschieden. Die Tabelle 8.41 gibt eine Übersicht über die Erfüllung der einzelnen Anforderungen.

	optional	unkritisch	kritisch	Summe
nicht erfüllt	3	0	0	3
teilweise erfüllt	0	1	3	4
vollständig erfüllt	2	2	29	33
Summe	5	3	32	40

Tabelle 8.41 Erfüllung der Anforderungen

Von den nicht optionalen Anforderungen hat die Projektgruppe alle beachtet und 31 von 35 vollständig umgesetzt (~89%). Da nur bei zwei der vier teilweise erfüllten Anforderungen eine sinnvolle Funktionalität verloren gegangen ist, kann sogar von einem Erfüllungsgrad von >94% gesprochen werden, wenn man diese Anforderungen als zur Hälfte erfüllt abschätzt.

Kapitel 9
Evaluation

In diesem Kapitel soll das System evaluiert werden. Dazu wird festgestellt wie gut die Steuerung der Komponenten unter verschiedenen Umständen agiert. Verfolgte Ziele sind dabei die Kostenreduzierung und die Eigenverbrauchsoptimierung (siehe UC-20 und EMS-15A). Im Folgenden wird zunächst die allgemeine Durchführung beschrieben und anschließend auf die konkret durchgeführten Evaluationen eingegangen.

9.1 Durchführung

Es soll zum Vergleich stets ein kompletter Tag simuliert werden, d.h. 24 Stunden bzw. 96 Cycles. Es werden pro Gruppe mehrere Läufe durchgeführt und zwischen diesen Läufen Parameter variiert. Aufgrund von Lizenzeinschränkungen muss dies in einer realen Stunde geschehen. Simuliert wird zur besseren Vergleichbarkeit an dem PG-Rechner mit den zur Verfügung stehenden Kopplern. Die Evaluation wird im Debug-Modus durchgeführt.

9.2 Auswertung

Zur Auswertung werden die tatsächlichen Ausgaben der Koppler z.B. mittels der Evaluations-GUI aus der Datenbank gelesen. Dabei sind mindestens die produzierte elektrische und thermische Energie der jeweiligen Komponenten relevant. Zusätzlich wird die elektrische und thermische Grundlast benötigt falls die Gesamtbilanz betrachtet wird. Aus diesen Werten können für jedes Intervall die Kosten und der Eigenverbrauchsanteil bestimmt werden. Hat z.B. in einem Intervall das BHKW 50 kWh produziert, der Kühlschrank 60 kWh verbraucht und die Grundlast beträgt

20 kWh, so ist der Eigenverbrauch 100%, da die kompletten produzierten 50 kWh verwendet werden können, und die Kosten für dieses Intervall entsprechen den Bezugskosten für (60+20-50) kWh = 30 kWh. So können für alle Intervalle der 24 Stunden die Kosten aufsummiert werden und Anhand der Summe die Qualität der Steuerung zwischen verschiedenen Läufen verglichen werden. Ähnlich kann auch der Eigenverbrauch mittels eines Durchschnitts betrachtet werden.

9.2.1 Ergebnisse

Es werden Messreihen für die elektrische und ggf. thermische Ausgabe der einzelnen EVS-Komponenten betrachtet. Weitere Werte der Komponenten werden im Einzelfall betrachtet. Zusätzlich können die Wertereihen der Grundlast, des Wetters und deren verrauschte Werte aus der Datenbank gelesen werden. Die Gesamtkosten für den Betrieb eines Gebäudes setzen sich aus den thermischen und elektrischen Kosten zusammen, diese können wie folgt berechnet werden. Wir verwenden n Komponenten und betrachten k Intervalle. Der Verkaufspreis ist p_v und der Ankaufspreis p_a. Der Ausdruck $we_{a,b}$ bezeichnet den elektrischen Ausgabewert der Komponente a im Intervall b. Zusätzlich bezeichnet $we_{g,b}$ die elektrische Grundlast im Intervall b. Die Summe für das Intervall b wird bezeichnet mit $we_{s,b}$.

$$we_{s,b} = \sum_{i=1}^{n} w_{i,b} + w_{g,b}$$

Die Kosten für den Ankauf von Strom werden mit p_a und der Preis für den Verkauf mit p_v bezeichnet. Die elektrische Kostenbilanz $kosten_e$ berechnet sich wie folgt:

$$kosten_e = \sum_{i=1}^{k} \begin{cases} we_{s,i} * p_v & we_{s,i} > 0 \\ we_{s,i} * p_a & we_{s,i} < 0 \end{cases}$$

Für die Berechnung der thermischen Kosten $kosten_t$ wird die maximale Leistung des BHKW $wt_{BHKW,max}$, die produzierte thermische Leistung des BHKW $wt_{BHKW,b}$ im Intervall b, die thermische Grundlast $wt_{g,b}$ im Intervall b, der Preis für die Erzeugung von thermischer Leistung durch nicht EVS-Komponenten p_{tex} und der Preis zum Betrieb des BHKW p_{BHKW}.

$$kosten_t = \sum_{i=1}^{k} wt_{BHKW,i} * p_{BHKW} + \begin{cases} (wt_{g,i} - wt_{BHKW,max}) * p_{tex} & wt_{g,i} > wt_{BHKW,max} \\ 0 & else \end{cases}$$

Diese können mit der von der Optimierung errechneten Fitness verglichen werden. Wenn die Prognosen den tatsächlichen Werten entsprechen, sollten die Fitness und die Energiebezugskosten identisch sein.

Unser System erlaubt zudem Angaben darüber, wie groß ein potentieller Energiespeicher ausfallen sollte, siehe Abschnitt 9.4.

9.3 Eingaben

Die Änderung der Parameter für die verschiedenen Simulationen erfolgt soweit möglich mittels Szenario- bzw. Runparameter. Können Parameter nicht auf diesem Weg geändert werden, so werden sie mittels Änderungen im Code für die jeweiligen Läufe eingefügt. Die Parameter der Komponenten werden im Rahmen des Basisszenarios ermittelt und während der Evaluation nicht weiter verändert, siehe Abschnitt 9.4.

9.4 Einheitliches Evaluationsszenario

Um die Ergebnisse der Evaluation in einen sinnvollen Kontext darstellen zu können, wurde für die Evaluation ein Evaluationsszenario festgelegt. Dafür wurde zunächst das Gebäude mit folgenden Größen bestimmt.

- Anzahl der Wohneinheit
- Jährliche Thermische Grundlast
- Jährliche Elektrische Grundlast
- Standort (Längengrad / Breitengrad)

Als Beispiel für das Gebäude diente hier das Hotel Antares in Oldenburg mit 56 Zimmern. Mit geschätzten Zimmergrößen wurde eine Wohnfläche von 2425m² ermittelt. Bei einem mittleren Wärmeverbrauch von 144 kWh/m2 und Stromverbrauch von 76 kWh/m2 ergaben sich für das Szenario entsprechende Zahlen (siehe [42]).

- Anzahl der Wohneinheit: 56
- Jährliche Thermische Grundlast: 349.200 kWh/a
- Jährliche Elektrische Grundlast: 184.300 kWh/a
- Standort: 8.2384 (Long.) / 53.1267 (Lat.)

Aus den gegebenen Gebäudegrößen konnten in Folge hierfür passende Gebäudekomponenten berechnet werden. Als Gebäudekomponenten werden dabei ein BHKW, eine PV-Anlage, eine Batterie und ein Kühlhaus betrachtet. Die Waschmaschine wurde als steuerbarer Verbraucher nicht eingebunden, da sie nur zufallsgesteuert arbeitet und im Bezug zur Größe des Gebäudes keine nennenswerte Rolle im Verbrauch spielt.

Zunächst wurde die Größe eines passenden BHKWs bestimmt. Hierfür wurde unter zu Hilfenahme eines Online-Rechners eine Jahresdauerlinie ermittelt, die in Abbildung 9.1 dargestellt ist. Sie stellt den kumulierten Leistungsbedarf des Gebäudes in Abhängigkeit von der jährlich benötigten Nutzungszeit dar.

Anhand dieser Jahresdauerlinie wird eine Schätzung der Größe des BHKWs vorgenommen. Es wurde mindestens eine Betriebsstundenzahl von 5000 Stunden im Jahr für das BHKW festgelegt um ein möglichst wirtschaftliches BHKW zu verwenden. Um dies zu garantieren ist eine BHKW-Größe von 34 kW thermisch eine

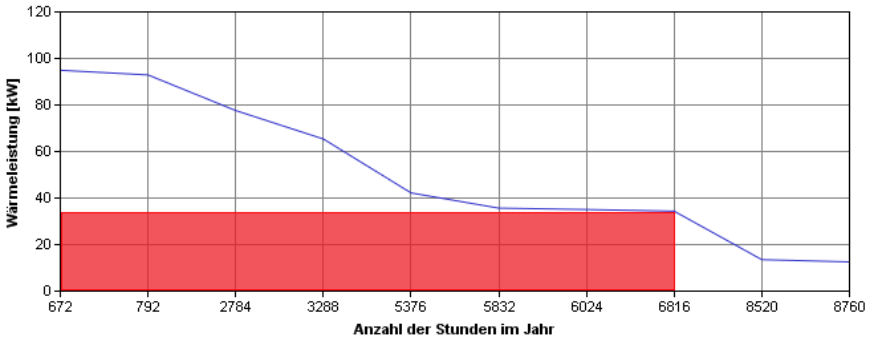


Abb. 9.1 Jahresdauerlinie (Stunden) der erforderlichen Heizleistung

passende Wahl. Mit dieser Größe kann fast eine Betriebsstundenzahl von 7000 Stunden im Jahr erreicht werden.

Als Beispiel für die benötigten Parameter diente das BHKW ASV 15/34 von Energiewerkstatt (siehe [9]). Dieses BHKW kann 34 kW thermisch und 15 kW elektrisch erzeugen. Um eine geeignete Pufferspeichergröße zu nutzen, haben wir uns nach den „Richtlinien zur Förderung von KWK-Anlagen bis 20 kW el“ des Bundesamt für Wirtschaft und Ausfuhrkontrolle (BAFA) gerichtet und eine Speichergröße von 1600 l als ausreichend erachtet.

Für die Größe der PV-Anlage ist man von einer Anlage auf dem Flachdach des Hotels ausgegangen. Die bebaubare Fläche von 70 m² wurde dafür komplett ausgenutzt. Die Daten des Solarmodul aleo S18J255 der aleo solar GmbH wurde als Grundlage verwendet (siehe [1]). 42 Module finden auf dem Flachdach Platz und kommen somit auf eine Maximalleistung von 7,8 kW-Peak.

Um eine passende Dimensionierung der Batterie für das Szenario zu erschließen, war vorgesehen einige Tagesläufe ohne eine Batterie durchzuführen. Durch den Vergleich von Stromerzeugung und -verbrauch sollte so die Batteriegröße bestimmt werden. Das Verhalten der Komponenten ließ solche Rückschlüsse auf die Batteriegröße leider nicht in einer optimalen Weise zu, so dass die Größe der Batterie selbst gewählt werden musste. Dabei ist man von 75% des kW-Peak der Solaranlage und zusätzlich 75% der maximalen elektrischen Leistung des BHKW für die Größe der Batterie ausgegangen. Diese Ergebnisse wurden noch aufgerundet auf eine Speicherkapazität von 20 kWh.

Unabhängig vom Rest des Gebäudes wurde ein Kühlhaus mit einem maximal Verbrauch von 0,7 kWh eingesetzt. Die genutzten Wetterdaten stammten aus dem Jahr 2014. Entsprechend dazu wurden Stromeinkaufs und -verkaufspreise in Höhe von 15,37 Cent/kWh und 3,35 Cent/kWh benutzt.

9.5 Testszenarien

Um die dargestellten Ziele adäquat und möglichst parallel evaluieren zu können, scheint es sinnvoll, die Evaluation auf drei Testszenarien aufzuteilen. Dies wird in den folgenden Abschnitten näher dargestellt.

9.5.1 Testszenario 1: Variation von Basisparametern

Um sinnvoll evaluieren zu können, ist zunächst Vorarbeit notwendig. Das erste Testszenario sieht daher vor, zu erarbeiten, wie viele Generationen in der Optimierung sinnvoll sind, um möglichst optimale Ergebnisse zu erzielen. Die Anzahl der Generationen hängt dabei von der Cycletime ab, d.h. der Zeit, die vom Taktgeber für einen Zyklus zur Verfügung gestellt wird.

Ebenfalls soll evaluiert werden, welchen Einfluss Zufallsauswahlen auf die Ergebnisse haben. Dazu werden verschiedene Runs mit gleichen Parametern, also z.B. den gleichen Wetterdaten und den gleichen Modellplarametern, jedoch unterschiedlichen Seeds verglichen. Die Seeds dienen dabei der Verrauschung im Simulationsframework, der Auswahl zufälliger Eltern in der Optimierung im Energiemanagementsystem sowie dem Sampling der Modelle auf den Kopplern.

Das wesentliche Ziel dieses Testszenarios ist die Vorgabe einer Cycletime, die von den anderen Testszenarien genutzt werden soll. Somit können die Testszenarien zwei und drei parallel evaluiert werden.

9.5.2 Testszenario 2: Variation von Optimierungsparametern

Das zweite Testszenario beschäftigt sich mit der Variation von Parametern, die in der Optimierung verwendet werden. Es soll die Rekombinationsstrategie untersucht werden. Dazu kann die Anzahl der Elternindividuen variiert werden sowie die Anzahl der daraus kombinierten Kinder. Ebenso kann die Mutationsrate variiert werden. Dies erlaubt eine Aussage darüber, welche Parameter im späteren Betrieb der Anwendung effizient sind und wie stark die Ergebnisse von den gesetzten Parametern abhängen. Zusätzlich wird die Prognose der Optimierung mit den Daten der Simulation verglichen, um zu prüfen, dass diese Werte sich ähneln.

9.5.3 Testszenario 3: Variation von Wetterdaten

Das dritte Testszenario beschäftigt sich mit der Frage, ob sich das System bei unterschiedlichen Wetterdaten stabil verhält und korrekt arbeitet. Dies kann mithilfe der

Variation der Wetterdaten erreicht werden. Dazu werden vier Durchläufe durchgeführt:

- **Simulation eines sonnigen Sommertags:** Es werden typische Wetterdaten eines Sommertags verwendet. Das Wetter des zugrunde liegenden Tages sollte überwiegend durch Sonnenschein bei wenig Bewölkung und Tagestemperaturen ab 15 °C gekennzeichnet sein.
- **Simulation eines sonnigen Wintertags:** Das Wetter des Wintertages sollte Temperaturen um den Gefrierpunkt aufweisen.
- **Simulation eines sonnigen Übergangstags:** Der Übergangstag sollte Temperaturen um 10 °C gekennzeichnet sein. Er stellt somit einen typischen Herbst- oder Frühlingstag dar.
- **Simulation eines bewölkten Übergangstags:** Außerdem soll ein stark bewölkter Übergangstag untersucht werden, um den Einfluss der Bewölkung nachvollziehen zu können.

Durch dieses Testszenario wird somit deutlich, in wie weit z.B. die Photovoltaikanlage durch das Wetter beeinflusst wird und welchen Einfluss das Wetter auf die Erträge hat.

9.6 Ergebnisse Testszenario 1: Variation von Basisparametern

Im Folgenden werden die Ergebnisse der Variation der Basisparameter dargestellt. Ziel ist die Ermittlung einer Cycletime, die eine optimale Anzahl an Generationen in der Optimierung produziert. Je länger die Cycletime, desto mehr Generationen sollten berechnet werden können. Diese Ermittlung ist wichtig, da die Optimierung einen genetischen Algorithmus verwendet und das Ergebnis der Optimierung daher von der Generationenanzahl abhängt (vgl. Abschnitt 5.10.3).

9.6.1 Untersuchung der Generationenanzahl bzw. Cycletime mit Zufallseinflüssen

Ziel des ersten Teils des Testszenarios war es zunächst, ein geeignetes Basisszenario samt geeigneter Basisparameter - etwa der Komponenten - zu entwickeln. Diese Ergebnisse und Parameter des Basisszenarios sind in Abschnitt 9.4 dargestellt.

Für ein Basisszenario ist es weiterhin wesentlich herauszufinden, wie viel Zeit die Optimierung bekommen soll, um eine Fitness zu ermitteln und die Samples auszuwählen. Dazu wurden verschiedene Durchläufe mit unterschiedlichen Cycletimes gestartet, deren Ergebnisse vergleichend beurteilt werden. Für eine erste Beurteilung wurden gleiche Seeds für die Optimierung und die Verrauschung der Wetterdaten verwendet. Für die gleiche Verwendung der Seeds der Komponenten müss-

Cycletime	5s	10s	15s	30s	60s
(1) durchschnittliche Anzahl an Generationen bis Fitness-Maximum	7,47	14,81	18,32	15,56	14,4
(2) Maximum der Anzahl an Generationen bis Fitness-Maximum	14	37	73	58	49
(3) durchschnittliche Anzahl berechneter Generationen pro Cycle	11,13	48,68	89,19	243,50	426,15
(4) Anzahl der untersuchten Cycles	96	96	96	96	55
(5) durchschnittliche Fitness in EUR	218,28	215,74	218,03	216,05	216,07
(6) Fitness Maximum in EUR	259,40	263,23	261,80	262,18	267,01
(7) Standardabweichung von (1)	2,632	8,344	13,538	9,203	9,955
(8) Standardabweichung von (5)	28,36	40,15	22,57	41,71	47,68
(9) Spannweite der jeweils besten Fitness über alle Cycles	127,62	231,78	120,78	241,35	233,17

Tabelle 9.1 Vergleich der erreichten Fitness und der Anzahl der Generationen von Durchläufen mit unterschiedlichen Cycletimes (mit Zufallseinflüssen)

ten diese durch eine technische Einschränkung vor jedem Durchlauf neugestartet werden - dies wurde im ersten Durchlauf nicht durchgeführt, sodass die berechneten Samples zwischen den unterschiedlichen untersuchten Cycletimes voneinander abweichen können. In einem späteren Schritt durch wurde diese technische Einschränkung durch einen Neustart der Komponenten umgangen, die Ergebnisse bei gleichen Komponenten- und Datenseeds werden in Abschnitt 9.6.2 dargestellt. Diese Ergebnisse sind in Tabelle 9.1 dargestellt.

Zunächst ist das insgesamt hohe Niveau der Fitness, welche die Schätzung der täglichen Erträge bzw. Kosten repräsentiert, auffällig. Maximalwerte (6) von bis zu 267,01 EUR pro Tag erscheinen unrealistisch. Dies ist auf eine falsche Verwendung von Einheiten zwischen der Optimierung und den Kopplern zurückzuführen und wurde später korrigiert (vgl.Abschnitt 9.6.2). Dennoch eignen sich die Werte aufgrund der relativ großen Spannweite (9) sowie der hohen Standardabweichung (8) (vgl. Tabelle 9.1) gut für die Untersuchung der Generationenanzahl, etwa um festzustellen, ob nach längerer Optimierungsdauer noch Verbesserungen erreicht werden. Außerdem verhält sich die Fitness jedoch für alle Cycletimes ähnlich, sodass mit der Evaluierung fortgefahren werden kann. Wird die durchschnittliche Fitness in EUR (5) betrachtet, zeigt sich, dass sich die Werte zwischen den unterschiedlichen Cycletimes nur geringfügig unterscheiden. Da diese Werte jedoch von den mithilfe der durch die Komponentenseeds berechneten Samples und damit vom Zufall abhängig sind, sollte kleinen Abweichungen nicht allzu große Bedeutung beigemessen werden.

Weiterhin ist zu erwähnen, dass aufgrund von Lizenzeinschränkungen die Anzahl der untersuchten Cycles (4) bei einer hohen Cycletime abweicht. Während 96 Cycles für die Simulation eines Tages notwendig sind, konnten bei einer Cycletime von

60 Sekunden lediglich 55 Cycles berechnet werden. Dies sollte bei Vergleichen berücksichtigt werden.

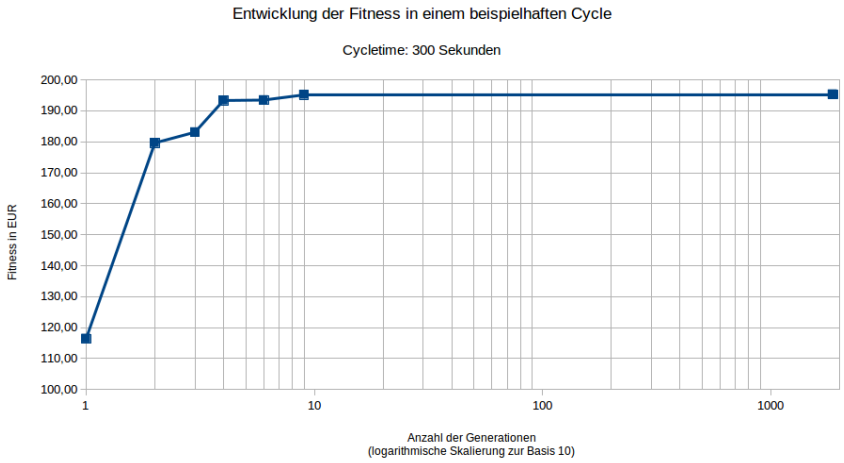


Abb. 9.2 Entwicklung der Fitness in einem beispielhaften Cycle bei einer Cycletime von 300 Sekunden

Kennzahl (1) gibt wieder, wie viele Generationen des evolutionären Algorithmus durchschnittlich notwendig waren, bis ein Maximalwert erreicht wurde. Wird zusätzlich die durchschnittliche Anzahl berechneter Generationen pro Cycle (3) betrachtet, ergibt sich, dass scheinbar nur wenige Generationen notwendig sind, um ein Optimum zu erreichen. Beispielsweise können bei einer Cycletime von 15 Sekunden durchschnittlich 89,19 Generationen berechnet werden, während bereits nach 18,32 Generationen ein Maximum erreicht wird. Die Anzahl der notwendigen Generationen weicht lediglich bei einer Cycletime von 5 Sekunden mit einem Wert von durchschnittlich 7,47 Generationen ab - dies kann als erstes Indiz gesehen werden, dass eine höhere Cycletime gewählt werden sollte: Die durchschnittliche Anzahl der berechneten Generationen reicht mit 11,13 nicht aus, um die Mittelwerte der notwendigen Generationen der anderen Cycletimes zu erreichen. In Abbildung 9.2 wurde ein Cycle eines Durchlaufes mit einer Cycletime von 300 Sekunden untersucht, um die These, dass nur wenige Generationen notwendig sind, genauer zu untersuchen. In dem Durchlauf wurden aufgrund der bereits dargestellten Lizenzeinschränkung nur 10 Cycles berechnet, von denen lediglich zwei ein Optimum nach mehr als 25 Generationen fanden. In dem in Abbildung 9.2 untersuchten Cycle wurde das Optimum in der 1868. Generation gefunden. Die Kurve verdeutlicht jedoch, dass sich die Fitness nur noch minimal geändert hat: Im Vergleich zu einem Wert, der bereits in Generation 9 ermittelt wurde, stieg die Fitness um lediglich 0,15 EUR. Eine derart lange Optimierungszeit ist demnach nicht lohnenswert.

Kennzahl (2) beschreibt den Maximalwert der Anzahl an Generationen aller durchlaufenen Cycles, welche für ein Fitness-Maximum notwendig waren. Bei einer

Cycletime von 15 Sekunden waren somit in einem Cycle 73 Generationen notwendig, um ein Optimum zu finden. In den anderen untersuchten Fällen war der Wert durchweg niedriger, auch wenn die Optimierung teilweise deutlich mehr Zeit hatte. Wird dementsprechend 73 Generationen als Zielwert der durchschnittliche Anzahl berechneter Generationen pro Cycle (3) genommen, wäre eine Cycletime von 15 Sekunden ausreichend um dieses Optimum zu finden. Bei Betrachtung der anderen, längeren Cycletimes stellt ein Wert von 73 Generationen bereits einen Ausreißer dar. Dies wird auch durch den Verlauf der Anzahl der Generationen bis zum Maximum, der in Abbildung Abbildung 9.3 beispielhaft für eine Cycletime von 15 Sekunden dargestellt wird, deutlich. Wenn zusätzlich die Standardabweichung der durchschnittlichen Anzahl an Generationen bis zum Fitness-Maximum (8) betrachtet wird, verdeutlicht sich das Bild: Die Anzahl der für ein Maximum benötigten Generationen schwankt nur gering um den Mittelwert (z.B. um 13,54 Generationen bei einer Cycletime von 15 Sekunden). Demnach wäre bereits eine Cycletime von 10 Sekunden ausreichend, um genügend Generationen für den Mittelwert zuzüglich der Standardabweichung zu berechnen. Um jedoch kleinere Ausreißer zu ermöglichen (bei Betrachtung der Abbildung Abbildung 9.3 wären dies Werte ab etwa 55 Generationen) empfiehlt es sich, eine Cycletime von 15 Sekunden, die etwa 89 Generationen pro Cycle ermöglicht, zu verwenden.

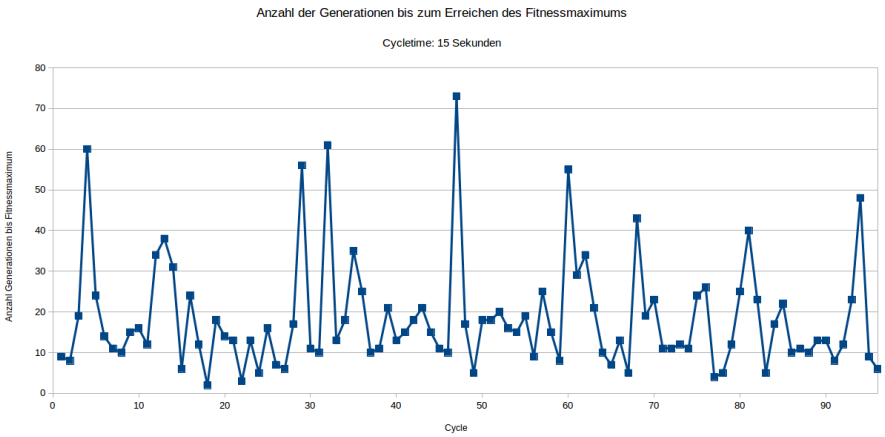


Abb. 9.3 Anzahl der Generationen bis zum Erreichen des Fitnessmaximums bei einer Cycletime von 15 Sekunden

9.6.2 Untersuchung der Generationenanzahl bzw. Cycletime ohne Zufallseinflüsse

In einer zweiten Untersuchung wurde durch einen Neustart der drei Koppler vor jedem Durchlauf sichergestellt, dass in den Modellen die Seeds korrekt verwendet werden und gleiche Zufallszahlen verwendet werden. Ferner wurde die Fitnessberechnung durch Angleichung der Einheiten in der Optimierung und den Modellen korrigiert.

Die Ergebnisse sind in Tabelle 9.2 zusammengefasst.

Zunächst lässt sich erkennen, dass trotz gleicher Seeds Zufallseinflüsse bestehen. Wäre dies nicht der Fall, sollte eine Erhöhung der Cycletime von 15 auf 30 Sekunden dazu führen, dass das Maximum der Anzahl an Generationen bis zum Fitnessmaximum (2) konstant bleibt oder sich erhöht. Die durchschnittliche Fitness (5) sollte ebenfalls konstant bleiben oder sich erhöhen, ggf. sollten sich höhere Maximalwerte der Fitness (6) bei höherer Cycletime zeigen. Dies ist nicht der Fall. Das lässt sich im Wesentlichen dadurch begründen, dass z.B. der *Random number generator* der Optimierung je nach Rechenzeit und -leistung im nächsten Zyklus einen unterschiedlichen Ausgangswert hat und dadurch andere Ergebnisse liefert. D.h. die Ergebnisse des *Random Number Generators* hängen von der Anzahl der bereits generierten Zufallszahlen ab. Das Zeitverhalten ist dementsprechend nicht deterministisch. Es wird dennoch untersucht, ob sich die Schlussfolgerungen aus den Ergebnissen denen aus Abschnitt 9.6.1 entsprechen.

Auffällig ist zunächst, dass die durchschnittliche Fitness (5) sowie die Maximalwerte der Fitness (6) ein deutlich realistischeres Bild ergeben. Prognostizierte Kosten von etwa -66,50 EUR am Tag erscheinen für das untersuchte Gebäude realistisch. Die Standardabweichung der Fitness (8) zeigt jedoch, dass die Fitness i.d.R. nur sehr wenig vom Mittelwert abweicht (um ca. 0,20 EUR). Dies wird auch durch die geringe Spannweite der jeweils besten Fitness über alle Cycles von etwa einem Euro bestätigt (vgl. Kennzahl (9)). In Abbildung 9.4 wird die Fitness im Zeitverlauf dargestellt. Die geringen Abweichungen lassen sich darauf zurückführen, dass ein Wintertag untersucht wurde, an dem die Photovoltaikanlage nur verhältnismäßig wenig produziert. Zudem war zunächst der Wechselrichter auf eine Leistung von 1000 Watt beschränkt, was eine maximale Arbeit von 0,25 kWh pro Viertelstunde zur Folge hatte. Dieser Parameter wurde im Szenario für weitere Untersuchungen auf 10000 Watt korrigiert.

Die durchschnittliche Anzahl berechneter Generationen pro Cycle (3) verhält sich wie zu erwarten ähnlich wie in Abschnitt 9.6.1. Die durchschnittliche sowie maximale Anzahl an Generationen bis zum Fitness-Maximum (Kennzahlen (2) und (3)) verhalten sich ebenfalls ähnlich. Die durchschnittliche Generationenanzahl ist in Abbildung 9.5 für eine Cycletime von 15 Sekunden dargestellt. Gegenüber der Untersuchung in Abschnitt 9.6.1 fallen die geringeren Schwankungen auf, die sich auch in einer niedrigeren Standardabweichung (7) widerspiegeln. Wichen die Generationen zuvor bei einer Cycletime von 15 Sekunden noch um durchschnittlich 13,54 Generationen vom Mittelwert ab, ist die Abweichung auf nun 8,05 Genera-

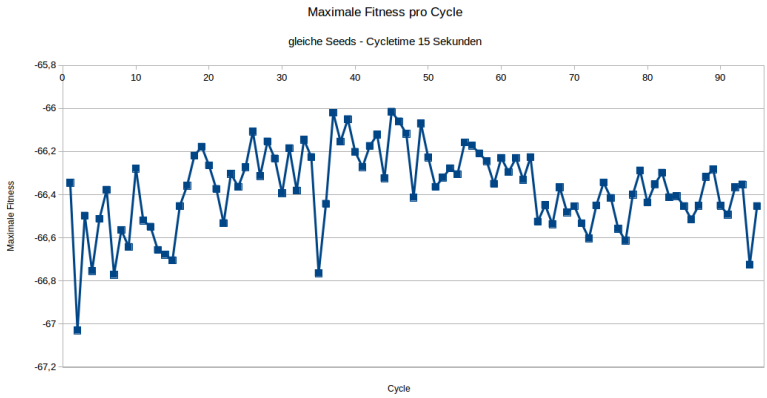


Abb. 9.4 Maximale Fitness pro Cycle bei einer Cycletime von 15 Sekunden (gleiche Seeds)

tionen gefallen. Bei Betrachtung einer Cycletime von 300 Sekunden ist ist zudem der geringe Maximalwert der Anzahl an Generationen bis zum Fitness-Maximum auffällig: Dies kann einerseits darauf zurückzuführen sein, dass lediglich 12 Cycles durchgeführt werden können. Andererseits führen die geringen Abweichungen der Fitness vielleicht dazu, dass auch nach längerer Zeit keine bessere Fitness gefunden werden kann. Dies kann auch als Erklärung für die geringe Standardabweichung dienen.

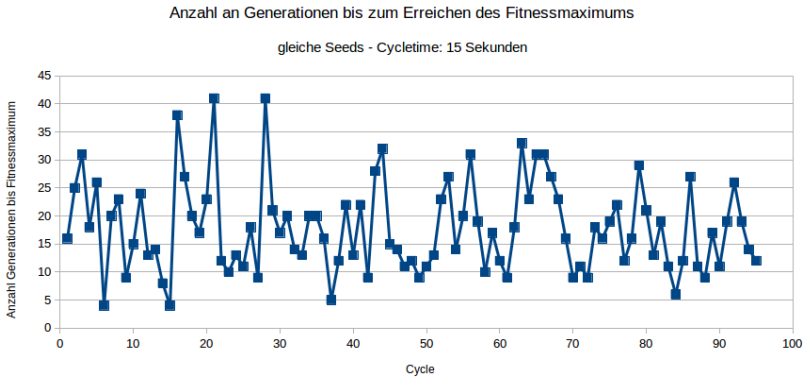


Abb. 9.5 Anzahl der Generationen bis zum Erreichen des Fitnessmaximums bei einer Cycletime von 15 Sekunden (gleiche Seeds)

Eine gemeinsame Betrachtung der durchschnittlichen Anzahl an Generationen bis zum Fitnessmaximum (1) sowie deren Standardabweichung (8) in Verbindung mit der Anzahl errrechneter Generationen (3) ergibt ebenfalls, dass theoretisch 10

Sekunden Cycletime ausreichend wären, um den Mittelwert zuzüglich der Standardabweichung zu errechnen. Das Maximum der Anzahl an Generationen bis zum Erreichen des Fitnessmaximums (3) spricht auch in diesem Durchlauf für eine Cycletime von 15 Sekunden: Nach spätestens 52 Generationen wurde ein Fitnessmaximum gefunden, während bei einer Cycletime von 15 Sekunden durchschnittlich 85,51 Generationen berechnet werden.

Cycletime	**5s**	**10s**	**15s**	**30s**	**300s**
(1) durchschnittliche Anzahl an Generationen bis Fitness-Maximum	8,34	16,17	17,94	17,42	19,67
(2) Maximum der Anzahl an Generationen bis Fitness-Maximum	12	46	41	46	52
(3) durchschnittliche Anzahl berechneter Generationen pro Cycle	11,10	49,35	85,51	201,78	2273,18
(4) Anzahl der untersuchten Cycles	96	96	96	96	**12**
(5) durchschnittliche Fitness in EUR	-66,61	-66,46	-66,37	-66,42	-66,28
(6) Fitness Maximum in EUR	-66,149	-66,020	-66,015	-66,042	-66,279
(7) Standardabweichung von (1)	2,289	7,94	8,046	9,256	11,949
(8) Standardabweichung von (5)	0,1987	0,2171	0,1876	0,1948	0,2087
(9) Spannweite der jeweils besten Fitness über alle Cycles	1,03	1,01	1,01	1,02	0,75

Tabelle 9.2 Vergleich der erreichten Fitness und der Anzahl der Generationen von Durchläufen mit unterschiedlichen Cycletimes (gleiche Seeds)

Auf eine erneute Untersuchung einer Cycletime von 60 Sekunden wurde aufgrund des geringen, zu erwartenden Verbesserungspotentials verzichtet. Insgesamt zeigt sich somit auch in diesem Durchlauf, dass eine Cycletime von 15 Sekunden empfehlenswert ist und die Ergebnisse aus Abschnitt 9.6.1 bestätigt werden.

9.7 Ergebnisse Testszenario 2: Variation von Optimierungsparametern

Die Evaluation der Optimierung soll untersuchen wie gut die Optimierung arbeitet. Dazu wird zunächst die berechnete Vorhersage der Optimierung mit den tatsächlichen Ausgabewerten der EVS-Komponenten verglichen. Anschließend wird untersucht welchen Einfluss einige Parameter der Optimierung haben.

9.7.1 Vorhersage Vergleich

In diesem Kapitel wird eine Vorhersage mit simulierten 24 Stunden bei aktiver Optimierung untersucht. Die Grundlast und die Wetterprognose werden während der Simulation nicht verändert. Es wird der Profit in Euro verglichen, d.h. die Summe der Kosten und Einnahmen aller EVS-Komponenten und der Grundlast. Die genauere Berechnung für die elektrischen und thermischen Kosten ist in Abschnitt 9.2.1 beschrieben.
Die Vorhersage für die folgenden 24 Stunden im ersten Intervall und die simulierten 24 Stunden sind in Abbildung 9.6 dargestellt, dabei stellt die X-Achse die Intervalle dar und der Y-Wert entspricht dem Profit in dem jeweiligen Intervall. Man erkennt, dass die Vorhersage durchgehend über der Simulation liegt. Um die Abweichung zwischen Vorhersage und Simulation genauer zu untersuchen ist die Differenz zwischen Simulation und Vorhersage in Abbildung 9.7 dargestellt, der Erwartungswert ist hier $0,50141$ und die Standartabweichung $0,02735$. Es ist also ein klarer Unterschied zwischen Vorhersage und Simulation zu erkennen, welcher etwas schwankend im Bereich von etwa 50 Cent in jedem Intervall liegt.

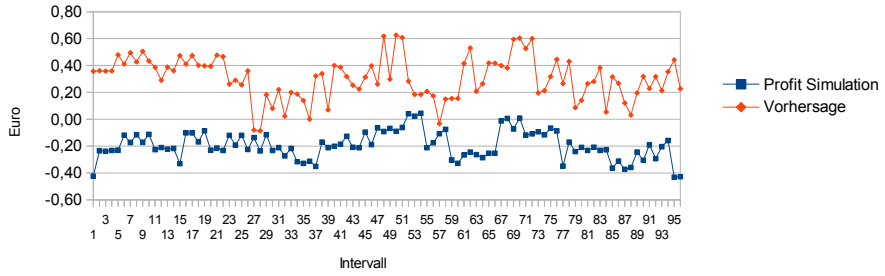


Abb. 9.6 Fitnessverlauf 24Stunden Vorhersage und Simulation

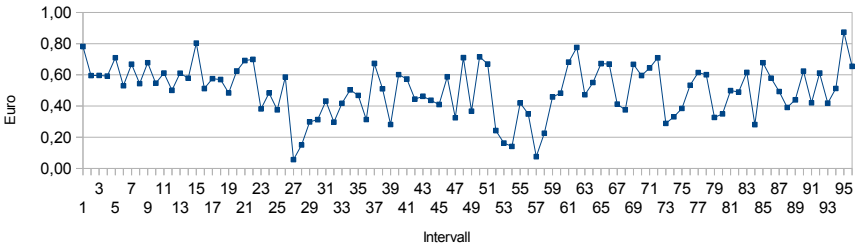


Abb. 9.7 Differenz zwischen 24Stunden Vorhersage und Simulation

Da die Vorhersage von 24 Stunden aufgrund des langen Vorhersagezeitraumes durchaus von den tatsächlichen Werten abweichen kann, wird ein kürzerer Vorhersagezeitraum untersucht. Dazu wird die Vorhersage für das jeweils folgende Intervall, also 96 Vorhersagen für jeweils 15 Minuten, mit dem simulierten Wert verglichen. Die Werte für einen Zeitraum von 24 Stunden sind in Abbildung 9.8 dargestellt. Die Differenz der Werte ist in Abbildung 9.9 dargestellt. Es ist zu erkennen, dass auch bei einem kurzen Vorhersagezeitraum eine deutliche Abweichnung zwischen Vorhersage und Simulation vorliegt, genauer ist der Erwartungswert $0,61044$. Allerdings ist die Schwankung deutlich geringer, was anhand der Standartabweichung von $0,00833$ erkennbar ist.

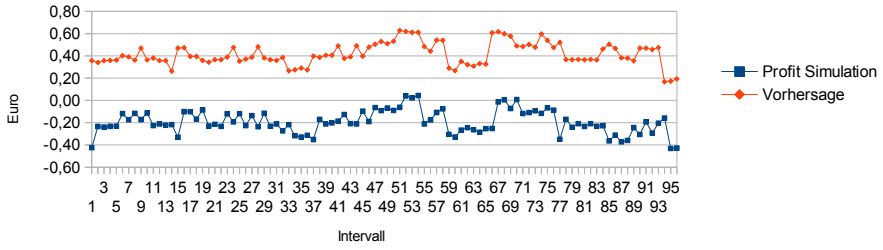


Abb. 9.8 Fitnessverlauf nächstes Intervall Vorhersage und Simulation

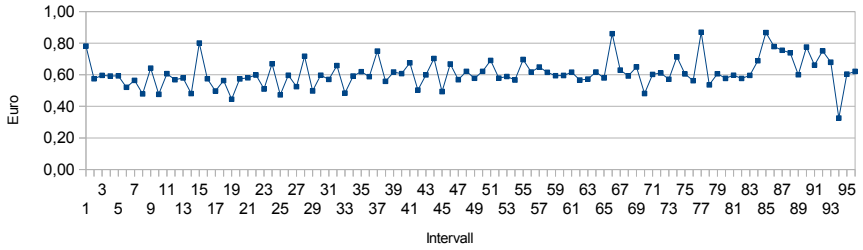


Abb. 9.9 Differenz zwischen nächstes Intervall Vorhersage und Simulation

Sowohl in der langfristigen als auch in der kurzfristigen Vorhersage weichen die Werte stark aber konsistent voneinander ab. Nach einer Untersuchung der Algorithmen wurde ein Fehler in der Fitnessberechnung der Optimierung gefunden und behoben.

9.7.2 Parameteruntersuchung

Der in der Optimierung verwendete genetische Algorithmus wird auf verschiedene Art durch Parameter in Geschwindigkeit und Stabilität (sodass bei finden einer guten Lösung keine viel schlechtere mehr auftritt) beeinflusst.
Grundsätzlich kann die der Funktion zur Verfügung gestellte Zeit verändert werden. Dies wurde bereits in der Basisevaluation (siehe Abschnitt 9) getan und wird deshalb an dieser Stelle ignoriert. Weiterhin kann die Population der evolutionären Strategie manipuliert werden, in dem die Größe der Eltern- und Gesamtpopulation angepasst wird. Abschließend kann die Mutation in Art und Häufigkeit angepasst werden.
Für diese Evaluation wurden zunächst die Populationsgrößen variiert, bevor der Einfluss der Mutationsrate beobachtet wurde.

9.7.2.1 Populationsgrößen

Als Ausgangspunkt wurde ein (15,100) Algorithmus verwendet. Dies bedeutet, das aus jeder Generation die 15 besten der 100 Kinder als neue Eltern ausgewählt werden. Dieser wurde dann durch Variieren der Eltern- und Kindanzahl zu (15,50), (15, 200) und (30,100) Algorithmen geformt, wobei jeweils die Anzahl der im 15-Sekunden-Intervall abgeschlossenen Generationen sowie die Fitness des Runs notiert wurden.

Generationsanzahl

Die Generationskurven je Population, welche in jedem Intervall die Anzahl der Generationen auftragen, sind in Abbildung 9.10, Abbildung 9.11, Abbildung 9.12 und Abbildung 9.13 zu finden. An diesen lässt sich erkennen, dass im Vergleich zum Ausgangsalgorithmus eine halbierte Generationsgröße auf 50 Kinder die Anzahl der abgeschlossenen Generationen etwa verdoppelt (vgl. und Abbildung 9.12), eine Verdopplung der Generationsgröße auf 200 Kinder diese Anzahl wiederum halbiert (vgl. und Abbildung 9.13). Dieser etwa lineare Zusammenhang entspricht den Erwartungen, da der Großteil der Berechnungen bei der Erzeugung der Kinder stattfinden, da in diesem Schritt bereits die Fitness des Individuums berechnet wird. Damit bleibt die Anzahl der Berechnungen pro Zyklusintervall (15 Sekunden) gleich.

Fitness

Weiterhin wurde die Fitness oben angesprochenen Runs mit verschiedenen Populationen ausgewertet.

Abbildung 9.14 zeigt die Fitnessverläufe der Runs in Abhängigkeit von den gewählten Parametern. Dabei lässt sich kaum ein Unterschied zwischen den Algorithmen feststellen.

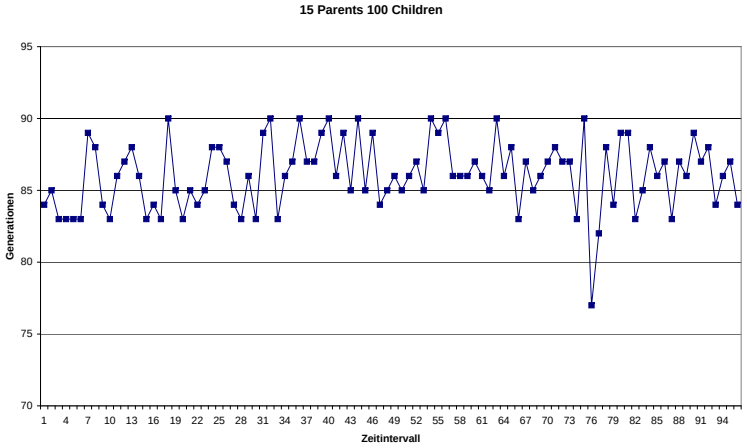


Abb. 9.10 Anzahl der Generationen für (15,100)

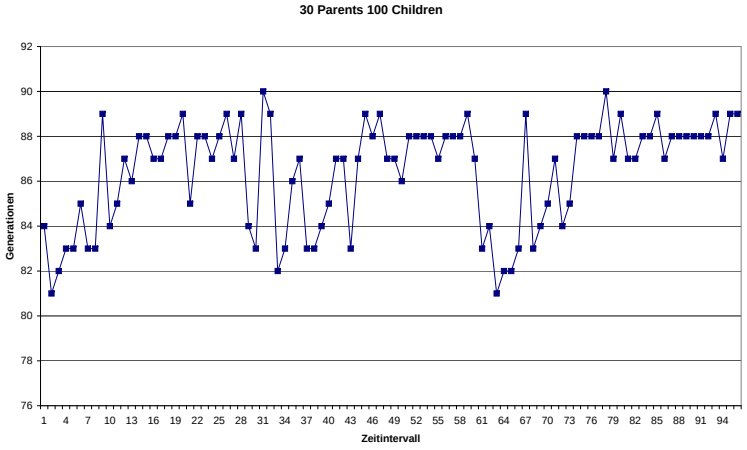


Abb. 9.11 Anzahl der Generationen für (30,100)

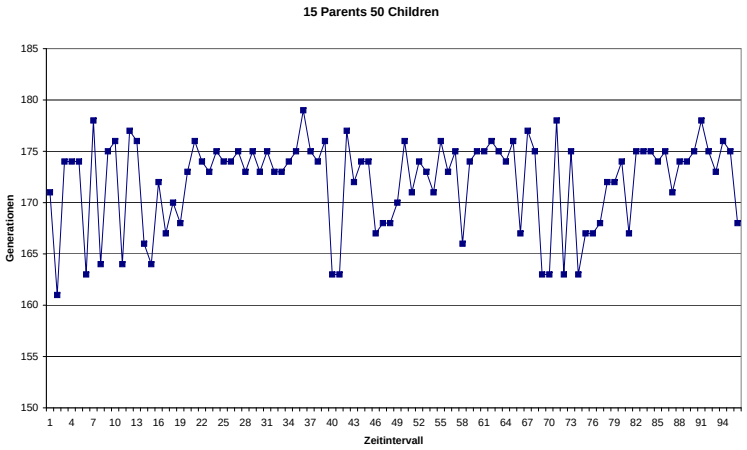


Abb. 9.12 Anzahl der Generationen für (15,50)

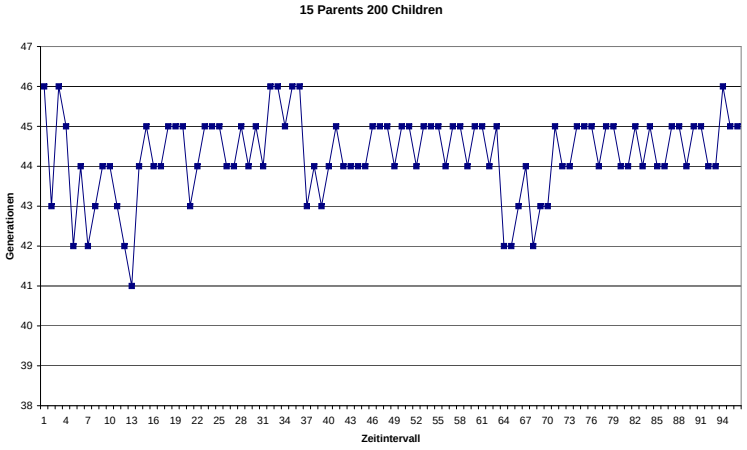


Abb. 9.13 Anzahl der Generationen für (15,200)

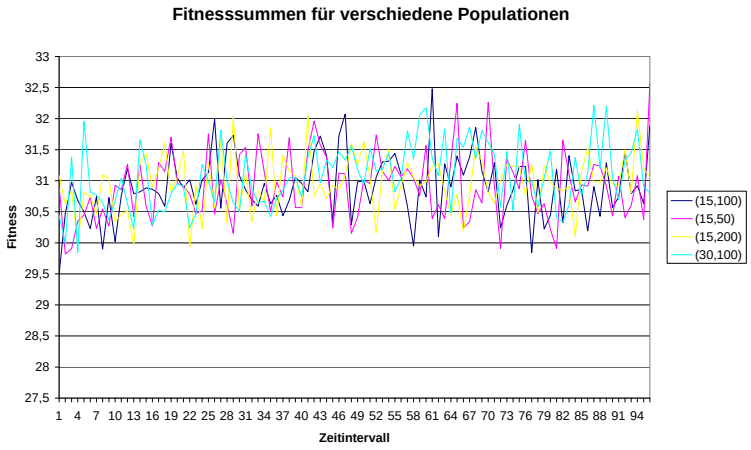


Abb. 9.14 Vergleich der Fitness der verschiedenen Algorithmen

Für eine genauere Betrachtung wurden deshalb die Mittelwerte und Standardabweichung berechnet.
Mittelwert:

- **(15,100):** 30,908
- **(30,100):** 31,075
- **(15,50):** 30,901
- **(15,200):** 30,963

Standardabweichung:

- **(15,100):** 0,513
- **(30,100):** 0,519
- **(15,50):** 0,546
- **(15,200):** 0,451

Es fällt auf, dass die Standardabweichungen sehr ähnlich ist, lediglich eine größere Population ist Vorteilhaft. Dies wird am Unterschied bei den Algorithmen (15,50) und (15,200) deutlich. Im Vergleich zur Fitness ist dies aber zu Vernachlässigen, es werden keine deutlich besseren Ergebnisse durch Populationsveränderungen erreicht.

9.7.2.2 Mutationsrate

Als Grundlage für die Evaluation wird der (15, 100) Algorithmus verwendet. Dieser Wird mit einer Mutationsrate von 25% ausgeführt.
Da dies bereits sehr hoch ist, werden Mutationsraten von 5% und 10% evaluiert.
Abbildung 9.15 zeigt die Verläufe der Fitnesssummen für die verschiedenen Mutationsraten. Die Anzahl der Generationen war von der Änderung der Mutationsrate wie erwartet nicht beeinflusst. Ein starker Unterschied der Fitness lässt sich allerdings wie bereits bei der Evaluation der Populationsgrößen nicht feststellen. Ein Blick auf die Mittelwerte und Standardabweichung bestätigt dies. Zu Vermerken ist, das 10% Mutationsrate besser erscheinen, wobei der Unterschied hier minimal ist und damit wahrscheinlich durch Zufall bedingt.
Mittelwert:

- **(15,100) bei 25%:** 30,908
- **(15,100) bei 10%:** 31.023
- **(15,100) bei 5%:** 30,882

Standardabweichung:

- **(15,100) bei 25%:** 0,513
- **(15,100) bei 10%:** 0,508
- **(15,100) bei 5%:** 0,564

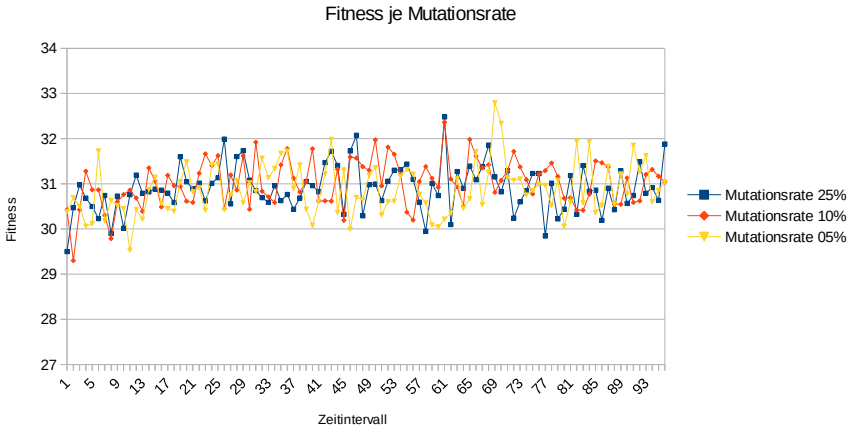


Abb. 9.15 Fitnesssummen für verschiedene Mutationsraten

9.7.3 Fazit

Im Rahmen der Evaluation der Optimierungskomponente wurde der Einfluss verschiedener Populationsgrößen sowie Mutationsraten untersucht.
Der von uns betrachtete Bereich ergab keine feststellbare Verbesserung der Optimierung. Keine der getesteten Variationen in Bezug auf Populationsgröße oder Mutationsrate konnte sich durch deutlich bessere Fitness absetzen. Festgestellt wurde allerdings ein merklicher Kostenunterschied zwischen Prognose und Simulation. Dies deutet auf einen Fehler in der Berechnung hin. Da dieser wie Abbildung 9.9 zeigt relativ konstant wirkt sollte er die weitere Untersuchung bzw. den relativen Unterschied der Evaluationsergebnisse nicht beeinflussen. Daher wird er an dieser Stelle ignoriert.

9.8 Ergebnisse Testszenario 3: Qualitative Analyse und Tests mit Wetterdaten

Ziel der qualitativen Analyse ist es, das Verhalten des Systems unter dem Einfluss verschiedener Wetterdaten zu untersuchen. Dabei liegt der Schwerpunkt auf Tests von korrektem Verhalten unter verschiedenen Bedingungen. Es folgt hier keine quantitative Analyse der Ergebnisse, da bei Einsatz in der Realität kein Einfluss auf das Wetter genommen werden kann. Es sollen vielmehr grundlegende Test durchgeführt werden, indem das Kapitel folgende Fragen klärt: Verläuft die Berechnung der Grundlast für thermische und elektrische Energie des simulierten Gebäudes für verschiedene Typtage korrekt? Wie arbeiten die EVS-Komponenten für die entsprechenden Tage? Weiterhin ist zu untersuchen, inwiefern die Wetterdaten das Erreichen des Optimierungsziels beeinflussen.
Auf diese Art wird das vollständige Systemverhalten getestet. Die Analyse stellt sicher, ob die eingegeben und generierten Daten korrekt an das Simulationsframework weitergegeben werden und die EVS-Komponenten die entsprechenden Eingabeparameter erhalten und sinnvolle Werte ausgeben. Ebenso wird durch diese Validierung sichtbar, ob Daten im EMS für die Testfälle korrekt verarbeitet werden.

9.8.1 Vorbedingungen

Um die Auswirkungen auf das System feststellen zu können, wurden Tage folgender Kategorien ausgewählt:

- Sommertag mit durchschnittlicher Temperatur von über 15 °C und Bedeckungsgrad < 5/8.
- Wintertag mit durchschnittlicher Temperatur von unter 5 °C und Bedeckungsgrad < 5/8.

- Übergangstag mit durchschnittlicher Temperatur zwischen 5 °C und 15 °C und Bedeckungsgrad < 5/8.
- Übergangstag mit durchschnittlicher Temperatur zwischen 5 °C und 15 °C und Bedeckungsgrad > 5/8.

Für die Kategorien wurden Wetterdaten aus dem Jahr 2014 ausgewählt und die Simulation mit dem einheitlichen Evaluationsszenario (siehe Abschnitt 9.4) durchgeführt.

9.8.2 Erwartungen

Für die Durchführung der Analyse des Systems werden folgende Erwartungen bezüglich der thermischen und elektrischen Grundlast formuliert:

- Die thermische Grundlast ist an einem Wintertag höher als an einem Übergangstag, dessen thermische Grundlast wiederum höher ist als die eines Sommertages.
- Die thermische Grundlast ist für Sommertage sehr gering.
- Die elektrische Grundlast unterscheidet sich für die jeweiligen Typtage nur in geringem Maße.

An das Verhalten des Blockheizkraftwerks mit Pufferspeicher werden folgende Erwartungen gestellt:

- Die Temperatur des Pufferspeicher des BHKWs verlässt nicht das im Szenario definierte, erlaubte Intervall von 45°C bis 90°C
- An Wintertagen läuft das BHKW häufig auf maximaler Stufe. Der Pufferspeicher dient häufig als Reserve zur Bereitstellung der benötigten thermischen Energie.
- An Sommertage wird das BHKW häufiger abgeschaltet, da der Bedarf an thermischer Energie sehr gering ist.

Für die PV-Anlage wird vom folgendem Verhalten ausgegangen:

- An heiteren Tagen wird deutlich mehr Strom durch die PV-Anlage erzeugt.
- An bewölkten Tagen ist die Stromproduktion sehr gering.
- Der Kurvenverlauf der erzeugten elektrischen Energie entspricht an sonnigen Tagen (minimaler Bedeckungsgrad) in etwa der Kurve einer Gauß-Verteilung.
- Schnelle Änderungen der Bewölkung sorgen für einen sprunghaften Kurvenverlauf der erzeugten elektrischen Energie.

Die Werte der Fitness der Optimierung werden durch nachstehende Erwartungen charakterisiert.

- Die Fitness ist abhängig von der Grundlast des Gebäudes und daher an Sommertagen höher als an Wintertagen.

9.8.3 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der vier Testdurchläufe dargestellt und anhand der Erwartungen untersucht. Dabei sind die dargestellten Graphen in fünf Kategorien eingeteilt. In Abbildung 9.16 bis Abbildung 9.19 sind die Temperaturverläufe der vier Tage dargestellt. Danach folgen in Abbildung 9.20 bis Abbildung 9.23 die Grundlastverläufe des Hotels und die erzeugte thermische und elektrische Energie des BHKWs. In Abbildung 9.24 bis Abbildung 9.27 werden die Temperaturen des Pufferspeichers angegeben und in Abbildung 9.28 bis Abbildung 9.31 die erzeugte elektrische Energie der PV-Anlage. Abbildung 9.32 bis Abbildung 9.35 veranschaulichen den Verlauf der Fitness für die vier Simulationen.

Temperatur

Die dargestellten Temperaturen entsprechen den von uns gewählten Tagen und wurden hier aufgeführt, um für die folgenden Analysen eine bessere Vergleichbarkeit zu erreichen. Da die Quelle der Daten nur Temperaturdaten bezüglich halber Stunden aufweist, ist die Temperatur für zwei Viertelstundenintervalle identisch.

BHKW und Grundlast

Die thermische Grundlast für die vier Testdurchläufe entspricht den genannten Erwartungen (siehe Abschnitt 9.8.2). Sie ist von der Größe im Sommer am niedrigsten und im Winter am höchsten. Interessanter ist hier das Verhalten der elektrischen Grundlast. Die Unterschiede über das Jahr sind doch unerwartet deutlich. So wird im Sommer maximal knapp über $2kWh$ pro Viertelstunde verbraucht. Am Wintertag sind es dagegen bis zu $24kWh$. Die Ursache liegt in der Berechnung der täglichen elektrischen Grundlast (siehe Gleichung 5.18). Der Einfluss des Faktors N_{WE} für die Anzahl der Wohneinheiten ist so groß, dass für die unterschiedlichen Typtage große Differenzen entstehen.
Weiterhin unerwartet ist die Grundlast für bewölkte und heitere Übergangstage (siehe Abbildung 9.22 und Abbildung 9.23). Gerade der Bedarf an elektrischer Energie ist für den bewölkten Übergangstag deutlich geringer als der für den heiteren Übergangstag. Der Grund dafür liegt in der unterschiedlichen Berechnung der täglichen elektrischen Grundlast für Wochen- und Sonntage. Der bewölkte Übergangstag aus der Testreihe befindet sich an einem Sonntag, wohingegen der sonnige Übergangstag auf einen Wochentag fällt.

Interessant ist, dass zwischen dem Verhalten des BHKWs bezüglich erzeugter thermischer Energie und der thermischen Grundlast kein erkennbarer Zusammenhang existiert. Besonders in Abbildung 9.23 beim bewölkten Übergangstag, wird zeitweise kaum thermische Energie erzeugt, obwohl eine sehr starke thermische Last vorliegt. Dies lässt vermuten, dass das erlaubte Temperatur-Interval (45 °C bis 90 °C) des Pufferspeichers verletzt wird, was jedoch nicht der Fall ist (siehe Abbil-

dung 9.27). Somit wird dieses Constraint nicht verletzt und der gewählte Verlauf scheint dementsprechend valide zu sein.

Pufferspeicher

Es lässt sich beobachten, dass bei allen Testdurchläufen das erlaubte Interval (45 °C bis 90 °C) eingehalten wurde. Dies ist verwunderlich, da auch bei großer Last zeitweise die Heizung kaum Energie produziert. Dies lässt vielleicht den Schluss zu, dass eine fehlerhafte Berechnung im Matlabmodell des BHKWs vorliegt, kann aber wegen fehlender Kenntnisse über die zugrunde liegenden physikalischen Berechnungen nicht gezeigt werden.

PV-Anlage

Das Verhalten der PV-Anlage, dargestellt in Abbildung 9.28 bis Abbildung 9.31, entspricht den dokumentierten Erwartungen. An heiteren Tagen ist die erzeugte elektrische Energie der PV-Anlage deutlich höher als an bewölkten Tagen. An einem heiteren Sommer- und Übergangstag erzielt sie eine Energie von über 1,7kWh. An einem heiteren Wintertag wird zur Mittagszeit knapp 1 kWh erzeugt. Deutlich weniger Energie wird an einem bewölkten Übergangstag erzeugt. Hier liegt das Maximum bei etwa 0.24 kWh. Einen sonnigen Tag veranschaulicht die Abbildung 9.30. Hier ist die Kurve der erzeugte Energie vergleichbar mit der Kurve einer Gauß-Verteilung mit einem Maximum zur Mittagszeit. Einen schnell wechselnden Grad der Bewölkung charakterisiert der Kurvenverlauf in Abbildung 9.28. Hier gibt es große Sprünge im Graphen, was darauf schließen lässt, dass zu den Zeitpunkten um etwa 13 Uhr und 15 Uhr der Bewölkungsgrad kurzzeitig enorm zugenommen hat.

Fitness

Es wurde davon ausgegangen, dass der Fitnesswert der Optimierung mit der elektrischen und thermischen Grundlast des simulierten Gebäudes korreliert. Die Ergebnisse (siehe Abbildung 9.32 bis Abbildung 9.35) der vier Testdurchläufe bestätigen die Erwartung. Für einen Sommertag wurde ein positiver Fitnesswert von etwa 30.30 € bis 31.80 € erzielt. Das bedeutet, dass mehr elektrische Energie eingespeist worden ist als vom Netz bezogen wurde. Für Übergangstage wurden Fitnesswerte von etwa -77 € bis -76.40 € für einen heiteren Tag und etwa −37.50 € bis −36.50 € für bedeckte Tage erreicht. Der Grund dafür liegt an den unterschiedlichen Einflussfaktoren für Wochen- beziehungsweise Sonntagen (siehe Abbildung 5.29). Der bedeckte Übergangstag ist ein Sonntag, weswegen die elektrische Energie vergleichsweise gering ist und somit eine bessere Fitness erzielt wurde. Der Fitnesswert für den heiteren Wintertag liegt zwischen etwa −106.50 € bis −105.80 €. Das entspricht den Erwartungen an das System, da die Grundlast für Wintertage am größten ist.

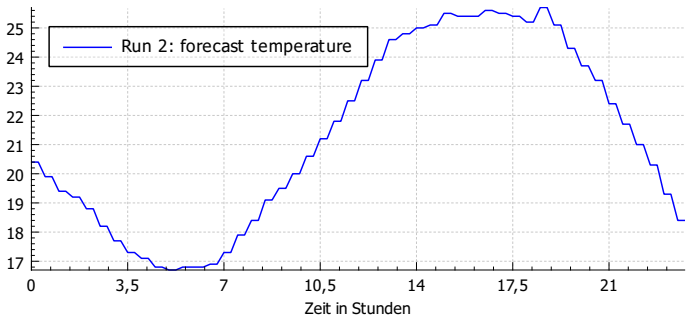


Abb. 9.16 Temperatur (heiterer Sommertag)

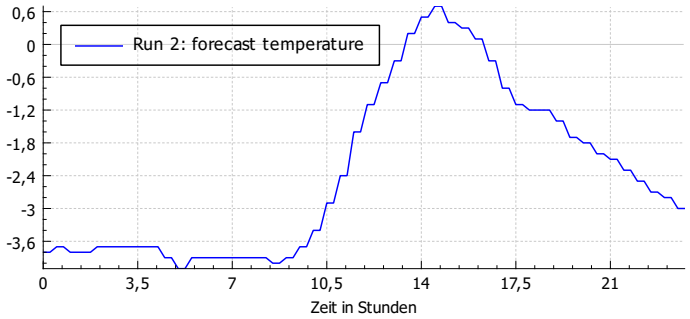


Abb. 9.17 Temperatur (heiterer Wintertag)

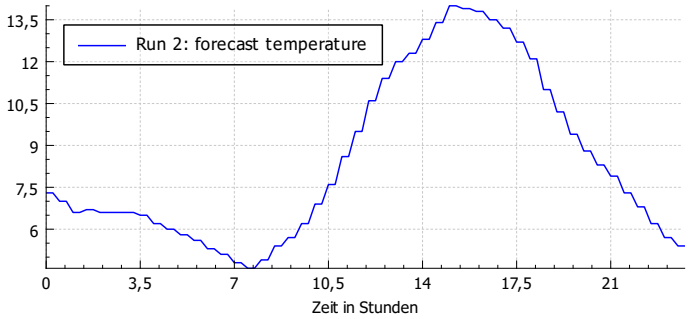


Abb. 9.18 Temperatur (heiterer Übergangstag)

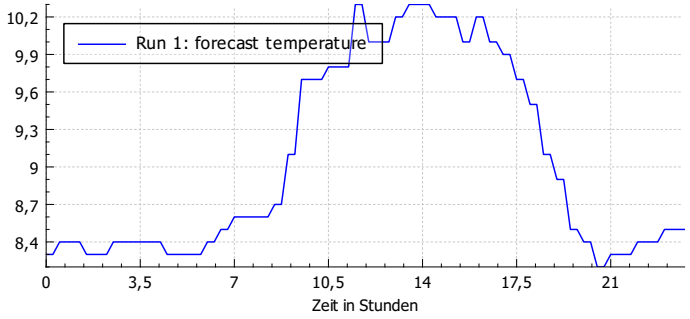


Abb. 9.19 Temperatur (bewölkter Übergangstag)

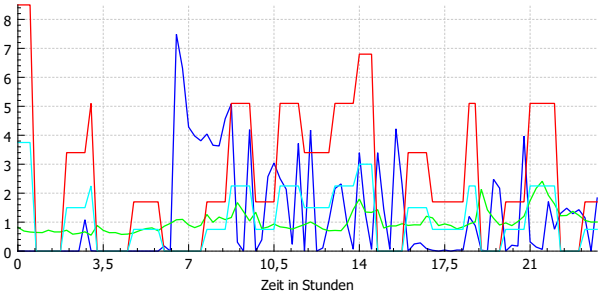


Abb. 9.20 heiterer Sommertag: blau = Wärmelast, grün = el. Grundlast, rot = BHKW Wärmeenergie, türkis = BHKW el. Energie (Daten in kWh)

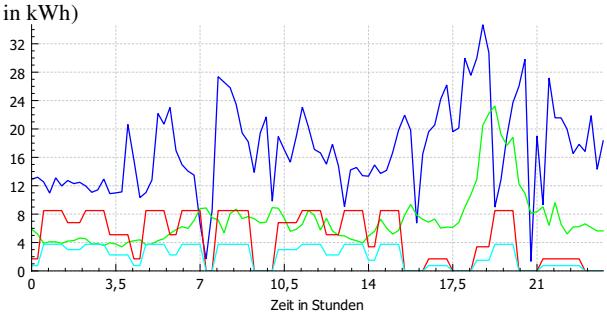


Abb. 9.21 heiterer Wintertag: blau = Wärmelast, grün = el. Grundlast, rot = BHKW Wärmeenergie, türkis = BHKW el. Energie (Daten in kWh)

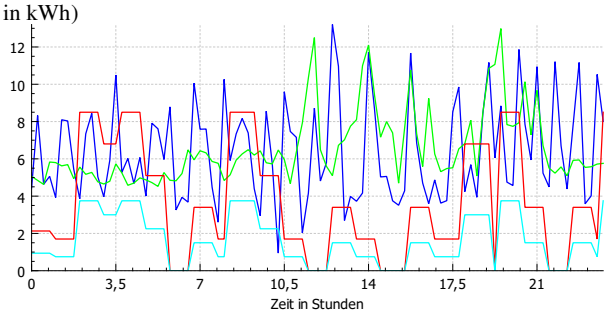


Abb. 9.22 heiterer Übergangstag: blau = Wärmelast, grün = el. Grundlast, rot = BHKW Wärmeenergie, türkis = BHKW el. Energie (Daten in kWh)

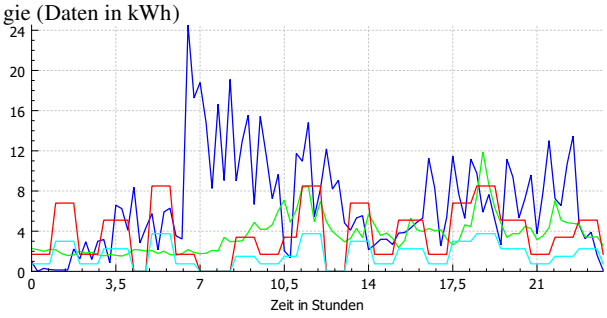


Abb. 9.23 bewölkter Übergangstag: blau = Wärmelast, grün = el. Grundlast, rot = BHKW Wärmeenergie, türkis = BHKW el. Energie (Daten in kWh)

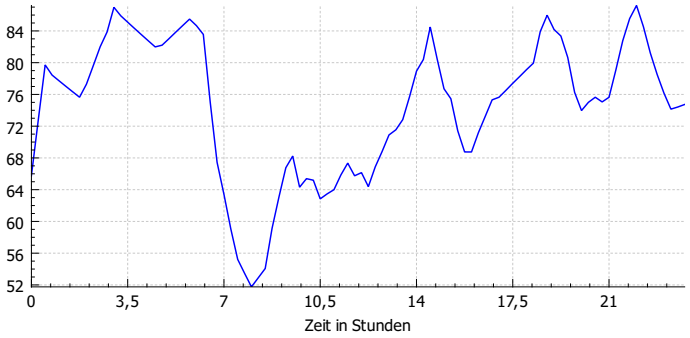


Abb. 9.24 Temperatur des Pufferspeichers in °C (heiterer Sommertag)

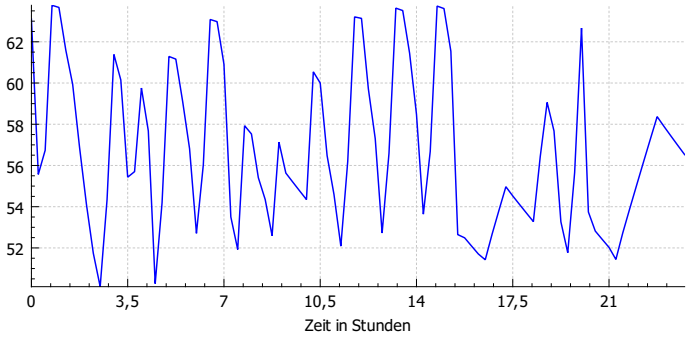


Abb. 9.25 Temperatur des Pufferspeichers in °C (heiterer Wintertag)

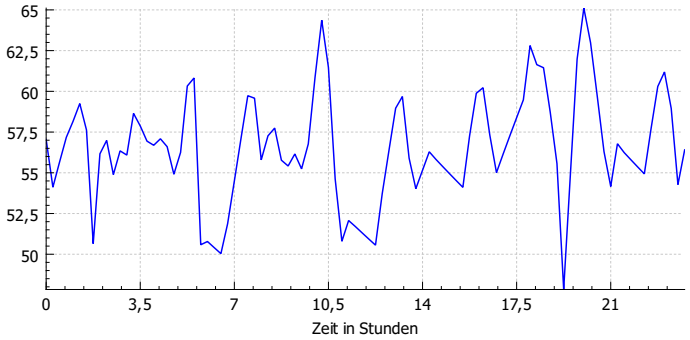


Abb. 9.26 Temperatur des Pufferspeichers in °C (heiterer Übergangstag)

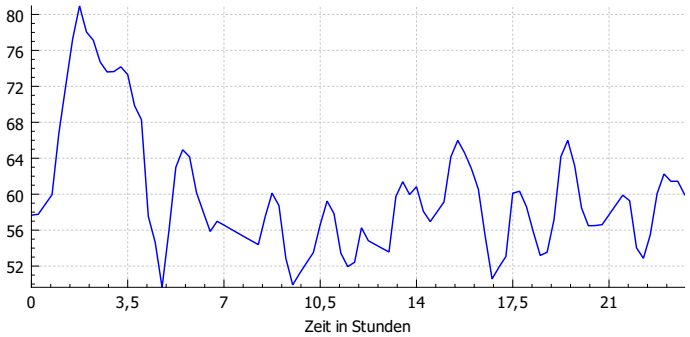


Abb. 9.27 Temperatur des Pufferspeichers in °C (bewölkter Übergangstag)

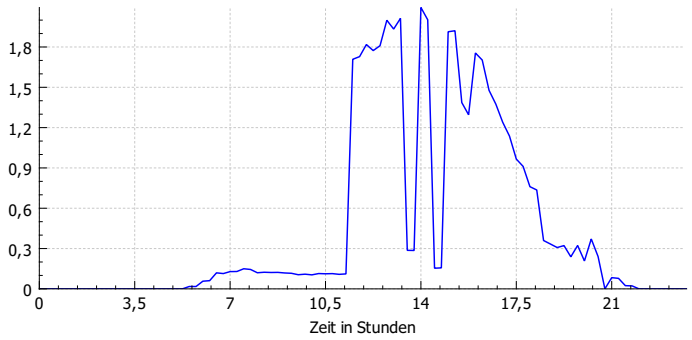


Abb. 9.28 Erzeugte el. Energie der PV-Anlage (in kWh) (heiterer Sommertag)

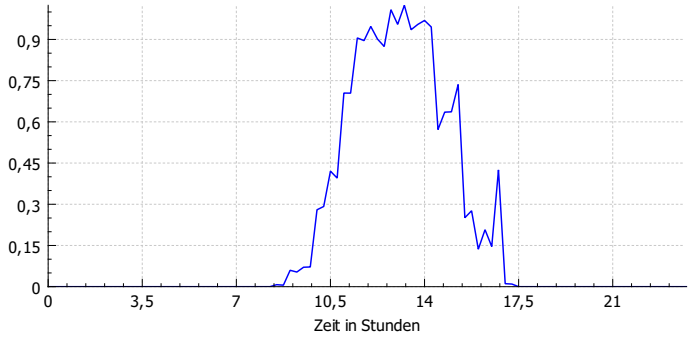


Abb. 9.29 Erzeugte el. Energie der PV-Anlage (in kWh) (heiterer Wintertag)

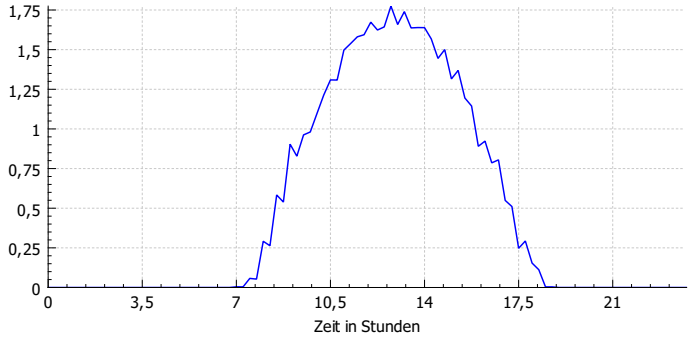


Abb. 9.30 Erzeugte el. Energie der PV-Anlage (in kWh) (heiterer Übergangstag)

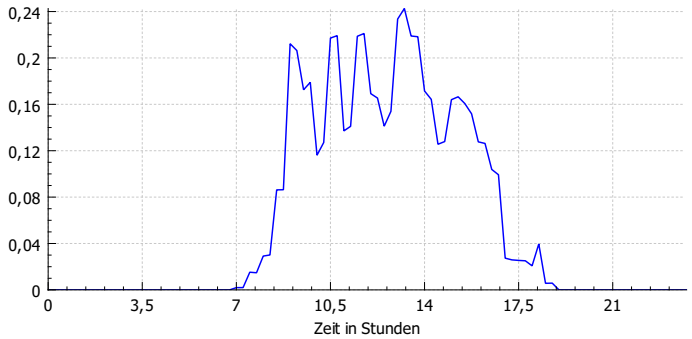


Abb. 9.31 Erzeugte el. Energie der PV-Anlage (in kWh) (bewölkter Übergangstag)

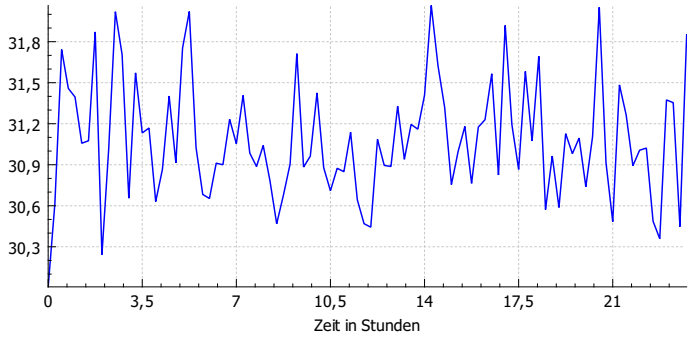


Abb. 9.32 Fitness der Optimierung für einen Tagesfahrplan in Euro (heiterer Sommertag)

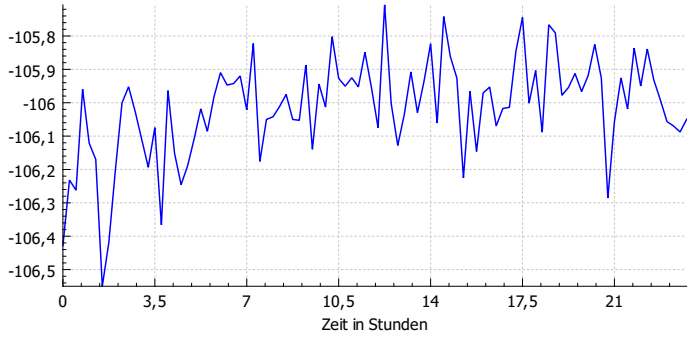


Abb. 9.33 Fitness der Optimierung für einen Tagesfahrplan in Euro (heiterer Wintertag)

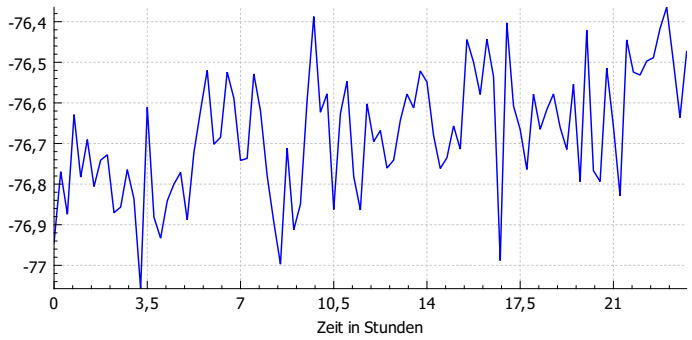


Abb. 9.34 Fitness der Optimierung für einen Tagesfahrplan in Euro (heiterer Übergangstag)

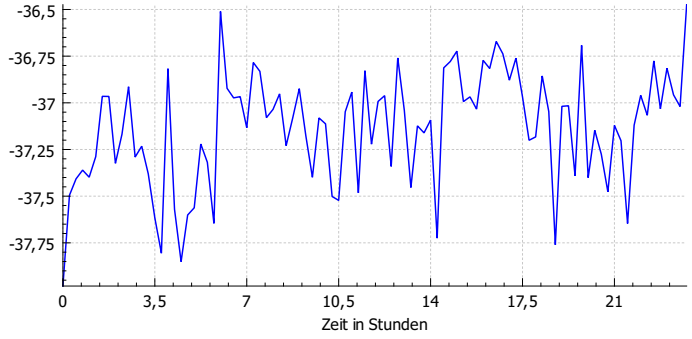


Abb. 9.35 Fitness der Optimierung für einen Tagesfahrplan in Euro (bewölkter Übergangstag)

Kapitel 10
Fazit

Als abschließendes Kapitel des Berichts folgt in diesem Abschnitt ein Fazit der Projektgruppe. Dafür werden in Abschnitt 10.1 zunächst die vergangenen zwei Semester reflektiert. Anschließend folgt in Abschnitt 10.2 ein Ausblick auf mögliche Verbesserung des Energiemanagementsystems, die im Rahmen der Projektgruppe nicht mehr leistbar waren, aber im Rahmen einer Fortsetzung sinnvoll und wichtig sind.

10.1 Reflexion

Zu Begin des Projekts gab es kaum Erfahrungswerte, wie ein Projekt dieser Größenordnung zu meistern ist. Zwar nahmen viele der Teammitglieder im Bachelorstudium am Softwareprojekt teil, jedoch ist dies kaum mit unserer vorliegenden Aufgabe zu vergleichen. Sowohl die technischen Anforderungen als auch die Organisation sind sehr verschieden. Während im Softwareprojekt von Anfang an die Meilensteine fest vorgegeben waren und nur die Zeit dazwischen ausgefüllt werden musste, war es während der Projektgruppe notwendig, alles in Eigenregie zu organisieren. Deshalb war die erste Aufgabe, sich einen Überblick über den Arbeitsauftrag und den Ablauf eines solchen Projektes zu verschaffen.

Dazu wurden zu Projektbeginn Arbeitsgruppen gebildet, in denen sich zu zweit oder dritt zu folgenden Themen, immer mit Bezug zur Projektgruppe, informiert wurde:

1. Projektmanagement
2. Anforderungsmanagement
3. Hardware
4. Software
5. Optimierungsstrategien
6. Dokumentation

7. Infrastruktur

Die Aufteilung erwies sich einerseits als sehr wertvoll, andererseits jedoch als problematisch. Mit dem Start des Projektes konnte sich die Gruppe insgesamt einen guten Überblick verschaffen, welche Themen voraussichtlich in dem Projekt bearbeitet werden müssen. Gleichzeitig konnten sich die Teams in das ihr zugewiesene Thema detailliert einarbeiten. Dies führte zu einer Konzentration von Expertise, sodass wir in unserem Team Ansprechpartner - *Experten* - für verschiedene Bereiche hatten. Dort liegt zugleich der größte Nachteil dieses Konzeptes: Da die Arbeitsgruppen später fast nahtlos in gleich bezeichnete Rollen übergingen, wurde das Expertenwissen der Personen immer größer, sodass Kopfmonopole entstanden. Gleichzeitig wurde es - nicht zuletzt bedingt durch das sehr umfangreiche System - immer schwieriger, einen Gesamtüberblick über das Projekt zu bekommen. Für ein optimales Risikomanagement wäre es daher sinnvoller gewesen, die Rollen regelmäßig zu tauschen. Der einmalige Rollentausch nach der ersten Projekthälfte hat jedoch gezeigt, dass ein solcher Tausch eine große Menge an Einarbeitungszeit erfordert. Diese stand nicht zur Verfügung, sodass kaum geeigneten Gegenmaßnahmen getroffen werden konnten. Auch wenn eine intensive Kommunikation zwischen den Projektmitgliedern bestand, konnten die Wissensmonopole nicht immer ausreichend ausgeglichen werden. Die Problematik dieser Monopole zeigte sich besonders in einer Phase (Abschnitt 2.3.4.2), als der Experte für die Aufgaben auf dem kritischen Pfad krank geworden ist. Dies hat zu erheblichen Verzögerungen in seinem Arbeitsgebiet geführt. Es ist allerdings fraglich, ob mit den zur Verfügung stehenden Ressourcen ein Auskommen ohne Wissensmonopole möglich gewesen wäre. Das Gesamtsystem ist sehr komplex sowie vielschichtig und erfordert an mehreren Stellen unterschiedliches Domänenwissen.

Die gemeinsame Erstellung eines umfangreichen Systems benötigt die im vorherigen Absatz erwähnte intensive Kommunikation untereinander, damit Gruppenmitglieder einen Überblick bekommen und sich die Arbeit eines Einzelnen in die richtige Richtung entwickelt. Die Möglichkeit dazu war geboten durch den Projektraum, der genutzt wurde, um die Arbeit zu koordinieren und die Fortschritte des Einzelnen in der Gruppe sichtbar zu machen.

Bereits seit Beginn des Projektes fanden zwei Mal die Woche Sitzungen statt, die einen Austausch und Beschlüsse ermöglichten. Dazu diente auch das im zweiten Halbjahr des Projektes eingeführte wöchentliche *Standup*, bei dem intern, d.h. ohne Betreuer, jeder Projektteilnehmer aufgefordert wurde, seinen persönlichen Fortschritt zu erläutern.

Neben den Sitzungen wurden Termine ohne Teilnahmepflicht vereinbart, die für regelmäßiges, gemeinsames Programmieren dienen sollten. Die beschlossenen Termine erreichten jedoch nach Abklingen einer anfänglichen Begeisterung im Allgemeinen nicht die gewünschten Teilnehmerzahlen, sodass ein verbindlicher Termin sinnvoller gewesen wäre. Aus Erfahrung der ersten Termine hätte dies ggf. zu einer schnelleren Problemlösung geführt, da auftretende Fragen und Problematiken schneller beantwortet werden könnten.

Wie in Abschnitt 2.2 erläutert, sind wir in unserem Projekt nach dem Vorgehensmodell Unified Process (kurz: UP) vorgegangen. Im Wesentlichen war der Einsatz

vom UP im Projekt auch sinnvoll. Es gab uns eine Richtung vor, was zu welcher Zeit in dem Projekt zu tun ist. Die Schwierigkeiten begannen jedoch bei der Anpassung des Vorgehensmodells auf unser konkretes Projekt. Dies beinhaltet unter anderem konkrete Tipps, die wir aufgrund der Projektstruktur nicht umsetzen konnten. Zum Beispiel sollten Tasks nach Möglichkeit an ca. einem Arbeitstag bearbeitet werden können. Dann wären tägliche Besprechungen sinnvoll gewesen, um immer auf dem aktuellen Stand zu sein. Diese beiden Tipps konnten wir nicht umsetzen, da die Aufgaben sonst viel zu kleinschrittig wären - die durchschnittliche tägliche Arbeitszeit liegt bei ca. 2,4 Stunden - und wir uns im Universitätsalltag nicht täglich gesehen haben. Stattdessen haben wir das erwähnte wöchentliche Standup durchgeführt.

Sinnvoll wäre zudem, die Iterationszeiten zu verkürzen. Dadurch, dass für die meisten Iterationen eine geplante Dauer von mindestens einem Monat vorgesehen war, konzentrierte sich die Arbeit teilweise auf das Ende der jeweiligen Iteration. Möglicherweise könnten eine Iterationsdauer von z.B. drei Wochen zu einem konstanteren Arbeitsverhalten führen.

Um die geleistete Arbeit zu reflektieren, wurde im zweiten Halbjahr der Projektphase zweiwöchentlich der prozentuale Fortschritt der jeweiligen Iteration mittels einer aus dem Ticketsystem exportierten Excel-Tabelle errechnet und den Betreuern vorgestellt. Diese Variante stellte sich als sehr hilfreich heraus. Einerseits ermöglichte sie dem Projektmanagement ein besseres Controlling, andererseits wurde dadurch jeder Projektteilnehmer aufgefordert, seinen Beitrag zum Gesamtfortschritt zu leisten. Zusätzlich wurde in regelmäßigen Gesprächen der Projektleitung die weitere Vorgehensweise erarbeitet und falls notwendig der Projektplan den aktuellen Gegebenheiten angepasst.

Vermutlich sinnvoller als diese Reflexion und das Controlling durch den Projektfortschritt wäre das Vorstellen der Ergebnisse nach jeder Iteration, wie in Abschnitt 2.2 erläutert. Leider ist dies nicht ohne Weiteres möglich gewesen. Gerade zu Beginn des Projekts lagen die Aufgaben in dem Entwickeln von Softwarebestandteilen, die ohne eine Integration nicht vorführbar waren, wie eine Zugriffsschicht auf die Datenbank, ein Optimierungsalgorithmus, das Deployen von Komponenten auf den Kopplern und ähnlichen Aufgaben. Möglicherweise wäre hier das Vorführen anhand von Testfällen oder ähnlichem sinnvoll gewesen. Auch wenn diese Maßnahme zeitliche Kapazitäten benötigt hätte und Zeit knapp war, so hätte sie sich durch erhöhte Motivation zur sorgfältigeren Ausführung der Aufgaben vermutlich ausgezahlt.

Erschwerend kommt hinzu, dass die Projektleiter in der Projektgruppe wenig Erfahrung in der Führung eines solchen Teams haben. Dies zeigt sich zum Beispiel darin, dass die Zeiten, wie lange einzelne Arbeitspakete zu bearbeiten sind, teilweise falsch eingeschätzt wurden. In die Planung flossen daher nach dieser Erfahrung die geschätzten Zeiten in bis zu vierfacher Höhe ein.

Darüber hinaus ist die Arbeitszeit, die jeder Einzelne in das Projekt einbringt nicht so gut vorherzusehen wie in einem Unternehmen, in dem die Mitarbeiter täglich zur Arbeit erscheinen. Im Rahmen der Projektgruppe herrschte eine freie Einteilung der Arbeitszeit, sofern die geforderten Stunden innerhalb einer Iteration erreicht wurden. Dies führt zu dem Phänomen, dass in einigen Zeiten mehr und in anderen, wie der Klausurenphase, weniger Arbeit verrichtet wird. Dadurch wird der geplante

Verlauf eines Projektes häufig durcheinander gewürfelt, was durch die schon oben erwähnte vielfältig notwendig Kommunikation verstärkt wird. Um diese Probleme etwas abzumildern, haben wir die an uns gestellten Anforderungen priorisiert. So wird gewährleistet, dass die Aufgaben, die für das Produkt unabdingbar sind, zuerst bearbeitet werden und wir schnellstmöglich ein lauffähiges Programm entwickeln. Außerdem wurden *Blocker* regelmäßig identifiziert und etwa durch die Einbeziehung weiterer Projektteilnehmer verstärkt angegangen. Dennoch hat vor allem die Verschiebung der Veröffentlichung der Alpha-Version (statt dem 15.11. konnte diese erst am 11.12. veröffentlicht werden) zu einigen Umplanungen geführt, was letztlich zu einer verkürzten Zeit für den Abschlussbericht sowie für die Evaluation führte. Für die Basisevaluation wären zudem mehr als die vorgesehenen zwei Personen sinnvoll gewesen, da Evaluation und Testphase fast verknüpft waren und eine Reihe von Bugs behoben werden mussten. Durch die Basisevaluation ist auch aufgefallen, dass beim Datenaustausch zwischen den Systemen und den EVS-Komponenten unterschiedliche Einheiten verwendet wurden, was Rechenfehler zur Folge hatte. Sinnvoll wäre gewesen, bereits zu Beginn Einheiten für die Interfaces festzulegen und zu dokumentieren.

Der Einsatz einer kontinuierlichen Integration, wie in Abschnitt 2.4 beschrieben, hat durchaus Vorteile. Dies hat sich vor allem bei der Integration des Zwischen- bzw. Abschlussberichts gezeigt. Durch Jenkins ist es schnell aufgefallen, falls noch ein Syntaxfehler in einem Latex Dokument war oder eine Datei vergessen wurde hinzuzufügen.

Allerdings muss für die kontinuierliche Integration Zeit für die Wartung und den Betrieb eingeplant werden. Dies zeigte sich bei den Jobs zum Bauen der verschiedenen Teilsysteme unseres Gesamtsystems. Ohne konsequentes Nachverfolgen und schnelle Wartung sinkt die Akzeptanz einer Integrationsumgebung sehr schnell und wird damit überflüssig, wenn nicht sogar eine Last. Erschwert wurde dies bei unserem Projekt dadurch, dass der Testrechner zugleich auch die Baujobs von Jenkins gehostet hat. Der Rechner war dadurch oft sehr stark ausgelastet, was zu weiteren Akzeptanzschwierigkeiten der Integrationsumgebung geführt hat.

Insgesamt haben haben wir - auch dank zahlreicher Programmier-Samstage - unser Ziel erreicht. Wir haben gezeigt, dass es möglich ist ein Energiemanagementsystem, wie in Abschnitt 1.1 dargestellt, zu Implementieren. Darüber hinaus haben wir sehr viel Neues gelernt und einige Herausforderungen gemeistert.

10.2 Ausblick

In unserem Projekt geht es in erster Linie darum, ein *Proof of Concept* für ein Energiemanagementsystem zu erstellen. Daher lag der Fokus der Projektgruppe auf dem Erstellen dieses Gesamtsystems. Die Bestandteile des Systems, also die Komponenten, die Simulation und die Optimierung, können vereinzelt verbessert werden.

So kann das Erstellen der Komponentensamples verbessert werden. Diese Samples stellen in unserer Anwendung ein notwendiges Mittel für eine möglichst realitäts-

nahe Optimierungskomponente dar, waren aber aufgrund zeitlicher Beschränkungen kein primäres Ziel. Die Erstellung und Verarbeitung kann noch verbessert werden. Zum Beispiel wurde die Frage, was passieren soll, falls in der vorgegebenen Zeit zu wenige Samples erstellt werden, lediglich theoretisch geklärt. Eine Implementierung war allerdings nicht notwendig, weil die Generierung der Samples in unserer Anwendung stets schnell genug war. Um einem Fehlverhalten der Optimierung vorzubeugen lassen sich zunächst zwei Fehlerfälle identifizieren. Zuerst könnte es vorkommen, dass weniger als die gewünschte Anzahl Samples generiert werden. Hierbei müsste die Optimierungskomponente mit dieser geringeren Zahl arbeiten. Weiterhin könnte kein einziges Sample fertiggestellt sein. Für diesen Fall ist es denkbar einen Buffer in die Samplingkomponente zu integrieren, der die Samples des vorherigen Zyklus speichert. Von diesem könnten dann der Optimierungskomponente Daten zur Verfügung gestellt werden, um trotz fehlender aktueller Datensätze noch eine annähernd sinnvolle Optimierung durchführen zu können.

Zusätzlich kann die Samplegenerierung als solche verbessert werden. Das Sampling des BHKW ignoriert zum Beispiel die Möglichkeit der Notabschaltungen, welche nach aktueller Umsetzung im Betrieb häufig eintreten und nicht wirtschaftlich sind. Weiterhin wurden im Sampling der Batterie mehrere Samplingstrategie eingebaut, konnten aber aus Zeitgründen in der Evaluation nicht bewertet werden. Eine Bewertung und Optimierung dieser und auch der Samplingstrategien der anderen Samplingkomponenten könnte zu einer Verbesserung des Gesamtsystems beitragen.

Um das Gesamtsystem für vielfältigere Zwecke einsetzbar zu gestalten, könnten weitere Optimierungsziele integriert und für den Anwender über die Steuerungsebene auswählbar gestaltet werden.

Da große Teile des Gesamtsystems die zentrale Datenbank verwenden, könnte die Implementierung durch die Verwendung eines objektrelationalen Mappings vereinfacht und beschleunigt werden. Dies ist erst im späteren Projektablauf deutlich geworden, nachdem beschlossen wurde, den Datenaustausch zwischen der Steuerungsebene und den übrigen Systemen über die Datenbank zu lösen.

Aufgrund der Annahme, dass die im Netzwerk befindlichen Rechner sich in einer Sicherheitszone befinden und die von der Kommunikation genutzten Ports von außerhalb nicht zugänglich sind, sollten für die Verwendung der Systeme in offenen Netzwerken Verschlüsselungsmechanismen integriert werden (siehe Abschnitt 6.3.1).

Grundlastdaten erhalten wir aus der Norm 4655 des VDI für Referenzlastprofile von Ein- und Mehrfamilienhäusern für den Einsatz von KWK-Anlagen. Wie der Name bereits andeutet, ist diese Quelle nicht für das von uns verwendete Szenario mit großen Gebäudetypen wie Krankenhäuser oder Hotels erstellt worden, was sich in extremen Schwankungen und Spitzen bei der Grundlastberechnung für entsprechend viele Wohneinheiten äußert. Daher sollte zukünftig eine passendere Quelle für Grundlastdaten gewählt werden.

Das Verrauschen der Datensätze geschieht in unserer Implementierung nach einem festen Prozentsatz. Das hat zur Folge, dass kleine Werte nur wenig, große Werte jedoch sehr stark verrauscht werden. Um das Verrauschen realistischer zu gestalten müssten die verschiedenen Datensätze in Abhängigkeit des Typs unterschieden werden, sodass beispielsweise für die Temperaturprognose eine maximale Abweichung

definiert wird und dementsprechend das Verrauschen unabhängig von der tatsächlichen Größe des Wertes geschieht. Für die Grundlastprognosen müsste ebenfalls eine maximale Abweichung definiert werden. Im System ist die Berechnung der Grundlast momentan auf die Klimzone TRY03 des DWD begrenzt. Dies sollte bei weiterer Verwendung erweitert werden.

Neben Anpassungen am Gesamtsystem wären weitere Tests mit umfangreicheren Szenarien oder anderen Komponenten sinnvoll, um die Einsetzbarkeit des Systems bei unterschiedlichen Szenarien zu überprüfen.

Glossar

A

Administrator (EMS) Der Administrator konfiguriert und wartet das Energiemanagementsystem (EMS). Die Konfiguration erfordert eine einfache und intuitive Bedienbarkeit des Systems. Zur Wartung benötigt er Einblick in Log-Dateien, die Informationen über den Systemzustand und der Kommunikation beinhalten. Gleichzeitig ist er Anwender des EMS und möchte die Steuerung mehrerer EVS-Komponenten in einem Gebäude optimieren. Damit er als Anwender den Erfolg und Stand der Optimierung verfolgen kann, muss er den Tagesverlauf der Optimierung und den Zustand der EVS-Komponenten einsehen können. Der Administrator besitzt Fachkenntnisse zu den EVS-Komponenten und technische Kenntnisse über das EMS und die Konfiguration des Systems. 46

Administrator (Simulation) Der Administrator des Simulationsframeworks erstellt das Szenario, für welches das Energiemanagementsystem (EMS) getestet werden soll. Um das Szenerio zu erstellen konfiguriert er die simulierten EVS-Komponenten und pflegt alle weiteren benötigten Daten ein. Hierfür wird eine einfache und intuitiv nutzbare Benutzeroberfläche benötigt. Des Weiteren wartet er das Simulationsframework. Dazu benötigt er einen Einblick in die Log-Dateien, die Daten zum Systemzustand, zur Kommunikation und der simulierten EVS-Komponenten. 46

Akteur Ein Akteur ist eine beteiligte Person, die sich außerhalb des Gesamtsystems befindet und als Benutzer beziehungsweise Anwender eines Systems oder des Gesamtsystems gilt. 40, 42, 59

B

Beckhoff Die Beckhoff Automation GmbH & Co. KG ist eine Unternehmen, welches Automatisierungstechnik basierend auf der Verwendung von PC-basierter Steuerungstechnik realisiert. 109

Bedeckungsgrad Der Bedeckungsgrad (oder auch Bewölkungsgrad) gibt an, wie groß der Anteil des Himmelskörpers ist, der mit Wolken bedeckt ist. Einge-

teilt ist der Bedeckungsgrad in Achtel. Er reicht von null Achtel (wolkenlos) bis acht Achtel (Himmel nicht erkennbar). 147, 196, 234

Blockheizkraftwerk Blockheizkraftwerke, kurz BHKW, sind Anlagen, die auf Basis von Verbrennungsmotoren oder Gasturbinen arbeiten und so elektrische Energie erzeugen. Zudem nutzen sie die bei der Erzeugung entstehende thermische Energie als Heizwärme. 40, 42, 103, 108, 109, 195, 235–237

Broker Der Begriff Broker bezieht sich auf die Systemkommunikation und bezeichnet eine Klasse, die für die Verteilung von Steuerbefehlen (z.B. zum Start der Optimierung) seitens der Steuerungsebene an das Energiemanagementsystem und das Simulationsframework zuständig ist. Gleichzeitig ist eine Rückmeldung des EMS oder des SF an die Steuerungsebene möglich. 126, 127, 131, 153, 154, 159

Bugfixing Bugfixing ist das Beheben eines Programmierfehlers. Betrieben wird das Bugfixing hauptsächlich während der Testphasen eines Projektes. Bugfixes nach dem bereits vollzogenen Release werden meist mit einem Patch oder erst der Installation eines neuem Release behoben. 196

C

Collector Der Begriff Collector meint im Rahmen der Systemkommunikation eine Klasse, die für das Sammeln und Verteilen von Logging-Nachrichten über Systemgrenzen hinweg zuständig ist. 131, 145, 158, 159, 163

Cycle Taktschritt bzw. Zyklus des Zeitgebers, der benötigt wird, damit die Systeme synchron funktionieren bzw. angesteuert werden können. 181, 186, 221, 222, 224

Cycletime Zeit in Sekunden, die der Taktgeber dem Simulationsframework und dem Energiemanagementsystem (gemeinsam) zur Verfügung stellt. Dort wird die Zeit z.B. für die Optimierung verwendet. 160, 219–226

E

Energiemanagementsystem Das Energiemanagementsystem ist ein System, dass das Zusammenspiel von Erzeugern, Verbrauchern und Energiespeichern hinsichtlich einer Zielfunktion über einen festgelegten Zeitraum optimiert. Das EMS enthält insbesondere das Optimierungssystem und die Auswertungskomponente. Das System kommuniziert mit den Komponenten über Steuersignale und Zustände. Für die Optimierung benötigt es Prognosen und aktuelle Daten. Das EMS soll so konzipiert werden, dass es für in der Realität existierende Gebäude, die EVS-Komponenten enthalten, eingesetzt werden kann. 43, 45, 47, 48, 59, 61, 63–70, 80–83, 96, 103, 104, 125, 138, 185, 186, 192, 193, 197–203, 205–210, 213

EVS-Komponente EVS-Komponenten sind die energieerzeugenden, -speichernden und -verbrauchenden Komponenten. Dazu gehören Photovoltaikanlagen, Blockheizkraftwerke, Batteriespeicher und steuerbare Verbraucher. 42, 49, 50, 54, 55, 63–68, 70, 73–75, 78–80, 82, 84, 85, 87, 96, 103, 104, 106–108,

135, 137, 181, 185–187, 194, 198–201, 203–206, 208, 210, 211, 213, 226, 234, 246

F

Fahrplan Das Energiemanagementsystem erstellt für alle EVS-Komponenten des zu simulierenden Gebäudes einen Fahrplan. Dieser bestimmt das Verhalten der EVS-Komponenten. 42, 55, 66, 199, 200

Fitness Als Fitness wird ein Wert bezeichnet, welcher zum Vergleich mehrerer Lösungen für das gleiche Problem dient und angibt, wie gut die jeweilige Lösung ist. Dieser Begriff wird vor allem im Bereich von genetischen Algorithmen und Heuristiken verwendet. 183, 220–222, 224

G

Gerätepool Der Gerätepool bietet dem Anwender die Möglichkeit, Geräte für die Konfiguration eines Szenarios auszuwählen. Zudem können neue Geräte dem Pool hinzugefügt oder bestehende Geräte bearbeitet bzw. gelöscht werden. 49, 84, 85, 211

Gesamtsystem Das Gesamtsystem besteht im Kontext dieses Projekts aus allen Systemen des Projekts. 91, 96, 151, 186, 194, 196

Grundlast Unter dem Begriff Grundlast wird im Rahmen des Projekts die elektrische und thermische Energie bezeichnet, die ein Gebäude, ausgenommen aller steuerbaren Verbraucher, verbraucht. 42, 64, 65, 109, 181, 198, 235

H

Hysterese Der Duden versteht unter Hysterese „das Zurückbleiben einer Wirkung hinter der sie verursachenden veränderlichen Kraft“. Bei einem Temperaturregler wird die Differenz zwischen Ein- und Ausschalttemperatur als Hysterese bezeichnet. 117

I

Initialplan Der Initialplan ist der erste Fahrplan, der mit dem Start der Optimierung erzeugt wird. Dieser ist abhängig von den Eigenschaften des Gebäudes (Grundlast, EVS-Komponenten) und Prognosedaten. Aufgrund von Abweichungen der Prognosedaten von tatsächlichen Daten müssen Änderungen am Initialplan vorgenommen werden. Die Evaluation des Energiemanagementsystems geschieht gegen den Initialplan. 65, 67, 68, 81

M

Modell View Controller MVC ist ein Entwurfsmuster zur Strukturierung der Benutzeroberfläche. Die Benutzeroberfläche wird in die drei Bereiche: Modell, View und Controller aufgeteilt. Dieses Entwurfsmuster hat den Vorteil, dass

die Bibliothek, mit der die GUI erstellt wurde leicht durch ein Anderes ersetzt werden kann, ohne den Funktionalen Quelltext neu schreiben zu müssen. 186

Modul Im Kontext unseres Projekts ist ein Modul ein elementares System mit einer Aufgabe. Ein Beispiel für ein Modul ist der Databaseaccesslayer. 31, 151, 196

O

Optimierung Die Optimierung ist die Hauptaufgabe des Energiemanagementsystems. Sie ist abhängig vom Optimierungsziel. Der Begriff umfasst sowohl die Erstellung des Fahrplans, als auch die Änderung beziehungsweise Anpassung des Fahrplans an aktuelle Gegebenheiten. 53–56, 63, 64, 67, 74, 80, 87, 198–201, 209, 210

Optimierungsschritt Ein Optimierungsschritt ist ein Schritt des Energiemanagementsystems, in dem der Fahrplan aktualisiert beziehungsweise angepasst wird. Die Dauer eines Optimierungsschritts beträgt 15 Minuten. In einer Simulation ist die Dauer eines Optimierungsschritts abhängig vom Zeitraffer. Ein Optimierungsschritt entspricht einem Zyklus. 68, 201

Optimierungsziel Das Optimierungsziel bestimmt die Zielfunktion, nach der die Optimierung durchgeführt wird. Beispiel für ein Optimierungsziel: Eigenverbrauchsoptimierung zur Minimierung der Energiebezugskosten. 53, 54, 64, 65, 198, 199, 234

P

PV-Anlage Eine Photovoltaikanlage, auch PV-Anlage, ist eine Anlage, die mittels Solarzellen Sonneneinstrahlung in elektrische Energie umwandelt. 40, 42, 108, 111, 235, 237

R

Run Ein Run bezeichnet eine Ausführung des Systems eines spezifischen Szenarios. 160, 185, 187, 188, 195, 219, 229

Rundaten Mit Rundaten werden alle zeitabhängigen Daten für eine Ausführung des Systems (Run) eines Szenarios bezeichnet. Hierunter fallen zum Beispiel Outputwerte für jede EVS-Komponente zu jedem Zeitschritt. 181, 185, 187, 189

S

Sample Ein Sample ist im Rahmen des Projekts ein möglicher valider Fahrplan, den eine EVS-Komponente zur Verfügung stellt. Dieser besteht aus Ausgabe- und Eingabeparametern, deren Werte durch den angebotenen Fahrplan bestimmt werden. 98, 181, 221

Sampling Der Prozess, der Samples generiert. 109, 135

Seed Zahl zum Initialisieren eines Zufallszahlengenerators. 181, 220, 224, 255, 256

Semantik Semantiken identifizieren den Inhalt von Parametern der EVS-Komponenten und den Datensätzen des Datenmodells. Stimmt die Semantik eines Parameters mit der eines Datensatzes überein, impliziert dies eine Zuordnung des Datensatzes zu dem entsprechenden Parameter. 124

Simulationsframework Das Simulationsframework dient in erster Linie dem Testen des Energiemanagementsystems. Es simuliert möglichst realitätsnah das Gebäude mit EVS-Komponenten, seiner Grundlast und die Umgebung, welche durch aktuelle Wetterdaten und Wetterprognosen definiert ist. 43, 44, 47, 48, 61, 64, 71–79, 96, 103, 123, 125, 127, 138, 159, 160, 185, 186, 192, 198, 203–208, 213, 234

Stakeholder Ein Stakeholder ist eine Person oder Gruppe, die in irgendeiner Weise vom Gesamtsystem betroffen ist und daher ein berechtigtes Interesse an der Entwicklung vorzuweisen hat. Er hat Einfluss auf die Anforderungen des Systems. 58–60

Steuerbare Verbraucher Steuerbare Verbraucher sind all jene Verbraucher (von elektrischer Energie) des simulierten Gebäudes, die nicht zur Grundlast des Gebäudes gehören und deren Steuerung vom Energiemanagementsystem übernommen werden soll. Ein Beispiel für einen steuerbaren Verbraucher ist eine Belüftungsanlage. 42, 109, 114

Steuerungsebene Die Steuerungsebene bildet die Schnittstelle zwischen den Anwendern und den zwei Systemen Energiemanagementsystem und Simulationsframework. Die Steuerungsebene ermöglicht das Konfigurieren der beiden Komponenten und koordiniert das Starten und Stoppen der Optimierung und Simulation, sowie die Einstellung für den Zeitraffer. Sie ermöglicht zusätzlich die Visualisierung der Zustände des Energiemanagementsystems und der Simulation. 47, 48, 61, 70, 80–87, 96, 125, 129–131, 145, 155, 156, 160, 185, 192, 198, 200, 202–204, 207, 209–213

Subsystem Ein Subsystem ist ein logisch abgegrenzter Teil eines Systems. Beispielsweise ist das Optimierungssystem ein Subsystem des Energiemanagementsystems. 151, 196

System Ein System ist im Kontext dieses Projekts ein logisch abgegrenzter Teilaspekt des Projekts, der für eine eigenständige Aufgabe zuständig ist. Das Projekt besteht aus den drei Systemen Energiemanagementsystem, Simulationsframework und Steuerungsebene. 31, 40, 42, 57–61, 92, 96, 125, 129, 136, 138, 151, 158, 159, 185–187, 192, 196, 207

Szenario Ein Szenario beschreibt eine Konfiguration, die für einen Simulationsdurchlauf genutzt werden kann. Zu einem Szenario gehören einerseits Daten über das Gebäude, wie die gewählten EVS-Komponenten, deren Kenndaten und deren Grundlast bezüglich Strom- und Wärmebedarf. Andererseits beinhaltet das Szenario Umgebungsdaten, die unter anderem Wetterdaten umfassen. 2, 42, 63, 64, 73, 74, 78, 80, 85–87, 104, 125, 137, 160, 181, 185, 187, 188, 197, 198, 204, 205, 208, 209, 211–213, 224

Szenariodaten Mit Szenariodaten werden alle zeitunabhängigen Daten eines Runs bezeichnet. Hierunter fallen zum Beispiel die enthaltenen Komponenten oder die Lage des Gebäudes. 181

T

TcCom Ein TwinCAT Component Object Model, kurz TcCom, definiert das Verhalten eines TwinCat-Moduls. Mit Hilfe des TwinCAT Target für Matlab-Simulink der Firma Beckhoff können aus Matlab-Modellen TcComs generiert werden. Diese TcComs können in der TwinCat-Laufzeitumgebung instanziiert werden und auf einem Koppler ausgeführt werden. 109

Test-Administrator Der Test-Administrator hat Zugriff auf den gesamten Umfang an Funktionalitäten, die das System bereitstellt. Er testet das gesamte System, um die Korrektheit und die Erfüllung der Anforderungen zu gewährleisten. 46

Typtag Ein Typtag dient der Kategorisierung eines Tages. Ein Tag unterscheidet sich in zugehöriger Jahreszeit (Winter-, Sommer- oder Übergangstag), der Kategorie Werk- oder Sonntag und dem Bewölkungsgrad (heiter oder bewölkt). Der Typtag bestimmt das für das Szenario ausgewählte Referenzlastprofil, mit dem der tägliche und viertelstündliche Bedarf an elektrischer und thermischer Energie für das simulierte Gebäude berechnet wird. 234–236

V

Visualisierungskomponente Die Visualisierungskomponente dient der Auswertung der Daten von Energiemanagementsystem und Simulationsframework, sowohl zur Laufzeit, als auch nach Ende der Simulation. 138

W

Wetterdaten Der Begriff Wetterdaten umfasst alle das Wetter und Klima betreffenden Daten, die Einfluss auf das System haben. Für das Projekt zum aktuellen Zeitpunkt relevante Daten sind die Zeitangaben Datum und Uhrzeit und zugehörige Angaben über Temperatur, Strahlung, Sonnenazimut und Bedeckungsgrad. Als Quelle dienen die Klimastation der Universität Oldenburg und die Klimadaten des DWD für Bremen. 146, 147

Literaturverzeichnis

1. ALEO SOLAR GMBH: *Solarmodul aleo S18*, März Juli 2014. http://www.aleo-solar.de/?id=27 (zuletzt abgerufen 18.03.2015).
2. BECKHOFF: *TwinCAT PLC – IEC 61131-3 Multi-PLC on the PC*. Website, 2014. http://www.beckhoff.com/english.asp?twincat/twincat_plc.htm (zuletzt abgerufen: 18.02.2015).
3. BREMER, JÖRG UND SONNENSCHEIN, MICHAEL: *Sampling the Search Space of Energy Resources for Self-organized, Agent-based Planning of Active Power Provision*. Environmental Informatics and Renewable Energies - 27th International Conference on Informatics for Environmental Protection, 2013.
4. BUSSKAMP, DIRK: *DBUS! Deutschland - Lizenzen*. Website, 2015. http://www.dbus.de/eip/kapitel04b.html (zuletzt abgerufen 02.02.2015).
5. BUKOLD, STEFFEN: *Die Kosten fossiler Energieimporte 2000-2012*. Studie, EnergyComment, 2013.
6. BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK: *Sichere Anbindung von lokalen Netzen an das Internet (ISi-LANA)*. Website, Januar 2015. https://www.bsi.bund.de/DE/Themen/Cyber-Sicherheit/ISi-Reihe/ISi-LANA/lana_node.html (zuletzt abgerufen 10.01.2015).
7. DUTCHGUILDER: *Iterative Development*. Picture, October 2007. http://commons.wikimedia.org/wiki/File:Development-iterative.png (zuletzt abgerufen 02.02.2015).
8. ECKERT, HOLGER: *Kältetechnische Planung*. PDF. http://www.kaelte-eckert.de/pdf/Kaeltetechnische_Planung.pdf (zuletzt abgerufen: 24.02.2015).
9. ENERGIEWERKSTATT GMBH & CO. KG: *Technische Daten des BHKW ASV 15/34*, März August 2014. http://www.energiewerkstatt.de/ (zuletzt abgerufen 18.03.2015).
10. ENGEL, CLAUS, ALEXANDER TAMDJIDI und NILS QUADEJACOB: *Ergebnisse der Projektmanagement Studie 2008 - Erfolg und Scheitern im Projektmanagement*. Präsentation, Dezember 2008. http://www.gpm-ipma.de/fileadmin/user_upload/Know-How/Ergebnisse_Erfolg_und_Scheitern-Studie_2008.pdf (zuletzt abgerufen 11.06.2014).
11. FOUNDATION, FREE SOFTWARE: *GNU General Public License*. Website, 2014. http://www.gnu.org/copyleft/gpl.html (zuletzt abgerufen 02.02.2015).
12. FOUNDATION, FREE SOFTWARE: *Warum man die Lesser GPL nicht für die nächste Bibliothek verwenden sollte*. Website, 2014. https://www.gnu.org/licenses/why-not-lgpl.de.html (zuletzt abgerufen: 02.02.2015).
13. FOWLER, MARTIN und MATTHEW FOEMMEL: *Continuous integration*. (Thought-Works) http://www.martinfowler.com/articles/continuousIntegration.html, 2006.

14. Free Software Foundation: *GNU Lesser General Public License*. Website, 2014. https://www.gnu.org/licenses/lgpl.html (zuletzt abgerufen 02.02.2015).
15. Gamma, Erich, Richard Helm, Ralph Johnson und John Vlissides: *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2008.
16. Gendreau, Michel, Jean-Yves Potvin, Michel Gendreau und Jean-Yves Potvin: *Handbook of Metaheuristics* -. Springer Science AND Business Media, 2. Aufl. Auflage, 2010.
17. GmbH, JULABO Labortechnik: *Betriebsanleitung Kompakt-Umlaufkühler AWC100*. PDF, 02 2011. http://www.julabo.com/sites/default/files/downloads/manuals/german/19504820.pdf (zuletzt abgerufen 10.03.2015).
18. Google Inc.: *Google C++ Style Guide*, November November 2014. https://google-styleguide.googlecode.com/svn/trunk/cppguide.html (zuletzt abgerufen 04.11.2014).
19. Gronau, Norbert: *Industrielle Standardsoftware: Auswahl und Einführung*. München [u.a.] : Oldenbourg, 2001.
20. Hampp, Tilmann und Markus Knauss: *Eine Untersuchung über Korrekturkosten von Software-Fehlern*. Softwaretechnik-Trends 28 (2), 2008.
21. Haucap, Justus und Beatrice Pagel: *Ausbau der Stromnetze im Rahmen der Energiewende : Effizienter Netzausbau und Struktur der Netznutzungsentgelte*. Ordnungspolitische Perspektiven, 55, 2014.
22. Heidinger, Roman: *Nutzung von Open Source Software in proprietären Softwareprojekten - eine Analyse aus urheberrechtlicher Sicht*. In: Armin B. Cremers (ed.), Rainer Manthey (ed.), Peter Martini (ed.) Volker Steinhage (ed.) (Herausgeber): *Informatik LIVE! Band 1, Beiträge der 35 Jahrestagung der Gesellschaft für Informatik e.V.*, Band P-67, Seiten 121–125, Bonn, 19. bis 22. September 2005. Gesellschaft für Informatik. http://cs.emis.de/LNI/Proceedings/Proceedings67/GI-Proceedings.67-24.pdf (zuletzt abgerufen 02.02.2015).
23. Hermann, Joachim und Holger Fritz: *Qualitätsmanagement. Lehrbuch für Studium und Praxis*. Carl Hanser Verlag, 1. Auflage, 2011.
24. Herrmann, Andrea und Knauss, Eric und Weissbach Rüdiger: *Requirements Engineering und Projektmanagement*. Springer Vieweg, 1. Auflage, 2013.
25. Hintjens, Pieter: *ZeroMQ - The Guide*. Website, September 2014. http://zguide.zeromq.org/page:all (zuletzt abgerufen 29.09.2014).
26. Hofmeyr, Neels: *CM Paper - Git oder Subversion?* (elego Software Solutions GmbH) http://www.elegosoft.com/files/Downloads/Publications/artikel_Git-vs-Subversion.pdf, 2013.
27. Hood, Colin und Wiebel, Rupert: *Optimieren von Requirements Management & Engineering*. Springer-Verlag, 1. Auflage, 2005.
28. iMatrix Corporation: *ZeroMQ Kommunikationsframework*. Website, September 2014. http://www.zeromq.org/ (zuletzt abgerufen 29.09.2014).
29. ITWissen Online-Lexikon für Informationstechnologie: *Webservice*. Website, Februar 2015. http://www.itwissen.info/definition/lexikon/Webservice-WS-web-services.html (zuletzt abgerufen 20.02.2015).
30. Jones, Capers und Olivier Bonsignour: *The Economics of Software Quality*. Prentice Hall, 1. Auflage, 2011.
31. Kainzbauer, Michael, Michael Kern und Bernhard Razocher: *Prozessmodell „Rational Unified Process“*. 2004. http://www.swe.uni-linz.ac.at/teaching/lva/ws03-04/se_uebung/05_gruppen/g1_ploesch/RUP_03.pdf (zuletzt abgerufen 12.03.2015).
32. Khan, Majid: *Easylogging++*. Website, September 2014. http://www.easylogging.org/ (zuletzt abgerufen 01.10.2014).
33. Konstantin, Panos: *Praxisbuch Energiewirtschaft: Energieumwandlung, -transport und -beschaffung im liberalisierten Markt*. Springer-Verlag Berlin Heidelberg, 3. Auflage, 1013.
34. Küveler, Gerd und Dietrich Schwoch: *Netzwerkprogrammierung mit Sockets*. In: *Informatik für Ingenieure*, Seiten 530–541. Springer, 2003.

35. Lexikon, Datenbanken Online: *ANSI-3-Ebenenmodell*. Website, 2012. http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/ANSI-3-Ebenenmodell (zuletzt abgerufen 26.03.2015).
36. Lipinski, Klaus, Hans Lackner, Oliver P. Laué, Gerhard Kafka, Alexander Niemann, Eberhard Raasch, Bernhard Schoon und Andrej Radonic: *CCCV (constant current constant voltage). IU-Ladeverfahren*. Website, November 2014. http://www.itwissen.info/definition/lexikon/CCCV-constant-current-constant-voltage-IU-Verfahren.html (zuletzt abgerufen 20.11.2014).
37. MariaDB Foundation: *MariaDB*. Website, März 2015. https://mariadb.org/ (zuletzt abgerufen 30.03.2015).
38. Martin Mikusz, Georg Herzwurm und: *Qualitätsmaßnahmen (konstruktiv/analytisch)*, 2014.
39. Mette, Anne und Jonassen Hass: *Guide to Advanced Software Testing*. ARTECH HOUSE, 1. Auflage, 2008.
40. Microsoft: *Dateitypen und Dateierweiterungen in Visual Studio*. Website, 2014. http://msdn.microsoft.com/de-de/library/xhkhh4zs%28v=vs.100%29.aspx (zuletzt abgerufen 29.09.2014).
41. Mitchell, John E.: *Branch-and-Cut Algorithms for Combinatorial Optimization Problems*. Handbook of Applied Optimization, Seiten 65–77, 2002.
42. MSc. Dipl.-Ing. Soara Bernard, Prof. Dr.-Ing. Karsten Voss: *Energieverbrauch in der Hotellerie*. DBZ Deutsche BauZeitschrift, DBZ Spezial 10-2012, Oktober 2012. http://www.enob.info/fileadmin/media/Publikationen/EnOB/Fachartikel/DBZ_Voss_Energieverbrauch_Hotels_pdf.pdf (zuletzt abgerufen 18.03.2015).
43. Musiol, Frank, Thomas Nieder, Thorsten Rüther und Peter Bickel: *Erneuerbare Energien im Jahr 2013*. Studie, Bundesministerium für Wirtschaft und Energie, 2014.
44. Nüttgens, Markus und Enrico Tesei: *Open Source - Produktion, Organisation und Lizenzen*. Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 157, Januar 2000. http://tesei.de/docs/iwi/heft157.pdf (zuletzt abgerufen 02.02.2015).
45. ORACLE Corporation: *MySQL*. Website, März 2015. http://www.mysql.de/ (zuletzt abgerufen 30.03.2015).
46. Pivotal Software Inc.: *RabbitMQ - Documentation*. Website, September 2014. http://www.rabbitmq.com/documentation.html (zuletzt abgerufen 29.09.2014).
47. PLCOpen: *TC1 - Standards*. Website, 2015. http://www.plcopen.org/pages/tc1_standards/ (zuletzt abgerufen: 24.03.2015).
48. Processes, Sustainable: *Smart Grid Use Case Management Process*. Entwurf, CEN-CENELEC-ETSI Smart Grid Coordination Group, 2012.
49. Ramey, Robert: *Boost C++ Libraries - Serialization Tutorial*. Website, 2014. http://www.boost.org/doc/libs/1_47_0/libs/serialization/doc/tutorial.html (zuletzt abgerufen 29.09.2014).
50. Rupp, Chris und SOPHIST GROUP: *Requirementsengineering und Management*. 4. Auflage, 2007.
51. Schmitt, Robert und Tilo Pfeifer: *Qualitätsmanagement. Strategien - Methoden - Techniken*. Hanser Verlag, 4. Auflage, 2010.
52. Solutions, Lytron Total Thermal: *Einflussfaktoren der Kühlleistung auf Rückkühlern*. Website, 2012. http://www.lytron.de/tools-technical/anwendungshinweise/kuehlungleistung-des-kuehlers (zuletzt abgerufen 10.03.2015).
53. Spillner, Andreas und Tilo Linz: *Basiswissen Softwaretest*. dpunkt.verlag, 4. Auflage, 2010.
54. Spillner, Andreas, Thomas Rossner, Mario Winter und Tilo Linz: *Praxiswissen Softwaretest*. dpunkt.verlag, 3. Auflage, 2011.

55. STROMVERBRAUCHINFO.DE: *Stromverbrauch von Waschmaschinen*. Website, November 2014. http://www.stromverbrauchinfo.de/stromverbrauch-waschmaschinen.phpl (zuletzt abgerufen 10.11.2014).
56. TESSENDORF, BERND: *GPL, LGPL, BSD, Shared Source: Was bedeuten die verschiedenen Lizenzmodelle?* In: THOMAS BEMMERL, RAINER FINOCCHIARO, ANDREAS JABS STEFAN LANKES MARTIN PÖPPE UNIV.-PROF. DR. HABIL. (Herausgeber): *Seminar Parallel- und Realzeitsysteme Sommersemester 2004*, Seiten 71–82. Lehrstuhl für Betriebssysteme, RWTH Aachen, Kopernikusstr. 16, 52056 Aachen, Germany, 22. und 23. Juli 2004. http://www.lfbs.rwth-aachen.de/papers/seminar/seminarband_2004SoSe.pdf#page=71.
57. THE QT COMPANY: *QT-Framework*, März 2015. http://www.qt.io/ (zuletzt abgerufen 24.03.2015).
58. UNICOMM: *Unicomm++ Kommunikationsframework*. Website, September 2014. http://www.libunicomm.org/ (zuletzt abgerufen 29.09.2014).
59. VDI-GESELLSCHAFT ENERGIETECHNIK: *Referenzlastprofile von Ein- und Mehrfamilienhäusern für den Einsatz von KWK-Anlagen*, 2007.
60. VERLAG FÜR DIE DEUTSCHE WIRTSCHAFT AG, VNR: *Comic zur Qualität*, August 2014.
61. W3C WORKING GROUP: *Web Services Architecture*. Website, Februar 2004. http://www.w3.org/TR/ws-arch/ (zuletzt abgerufen 20.02.2015).
62. WINKELHOFER, GEORG: *Management und Projektmethoden: Ein Leitfaden für IT, Organisation und Unternehmensentwicklung*. Springer-Verlag, 3. Auflage, 2005.
63. WIRTH, HARRY: *Aktuelle Fakten zur Photovoltaik in Deutschland*, 2014.
64. ZANK, INGO: *Sensibilität, Virtualität und Variabilität - Woran IT-Projekte scheitern!* Website, November 2009. http://www.ikmt.de/forum/showthread.php?tid=334&pid=462#pid462 (zuletzt abgerufen: 11.06.2014).
65. ZUSER, WOLFGANG, THOMAS GRECHING und MONIKA KÖHLE: *Software Engineering mit UML und dem Unified Process*. Pearson Studium, 2. Auflage, 2004.

Anhang A
Anhang

A.1 Sequenzdiagramme

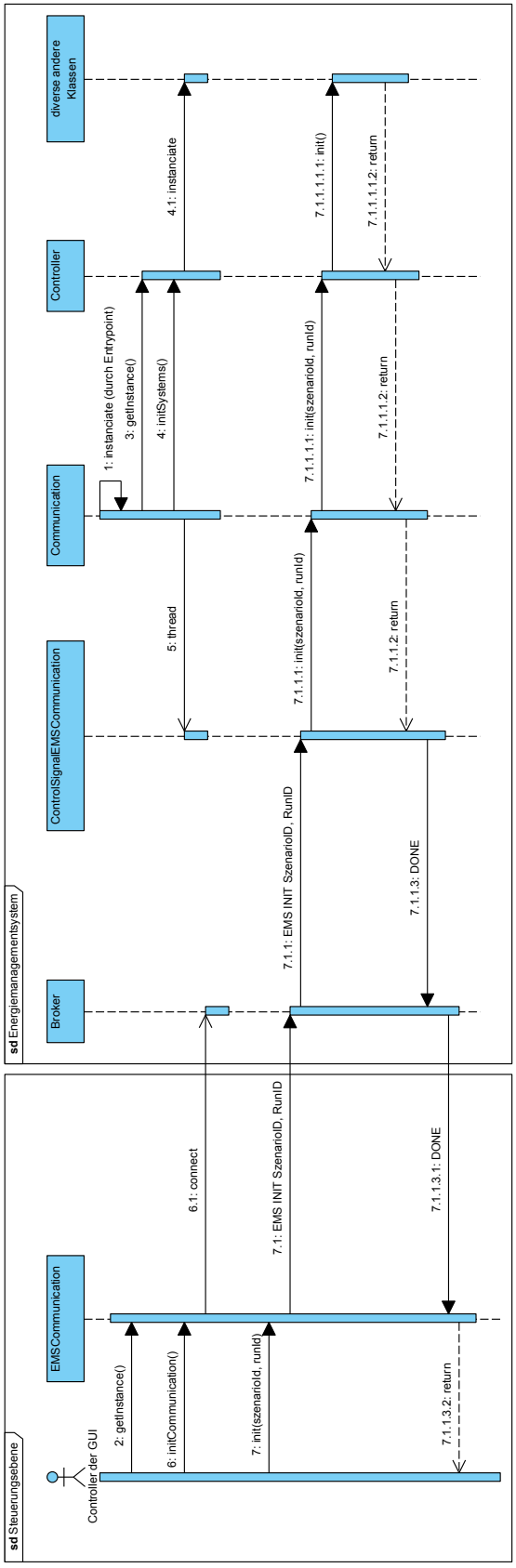


Abb. A.1 Sequenzdiagramm: Verbindungsaufbau der Steuerungsebene mit dem EMS und senden eines INIT-Signals (Version 1.0)

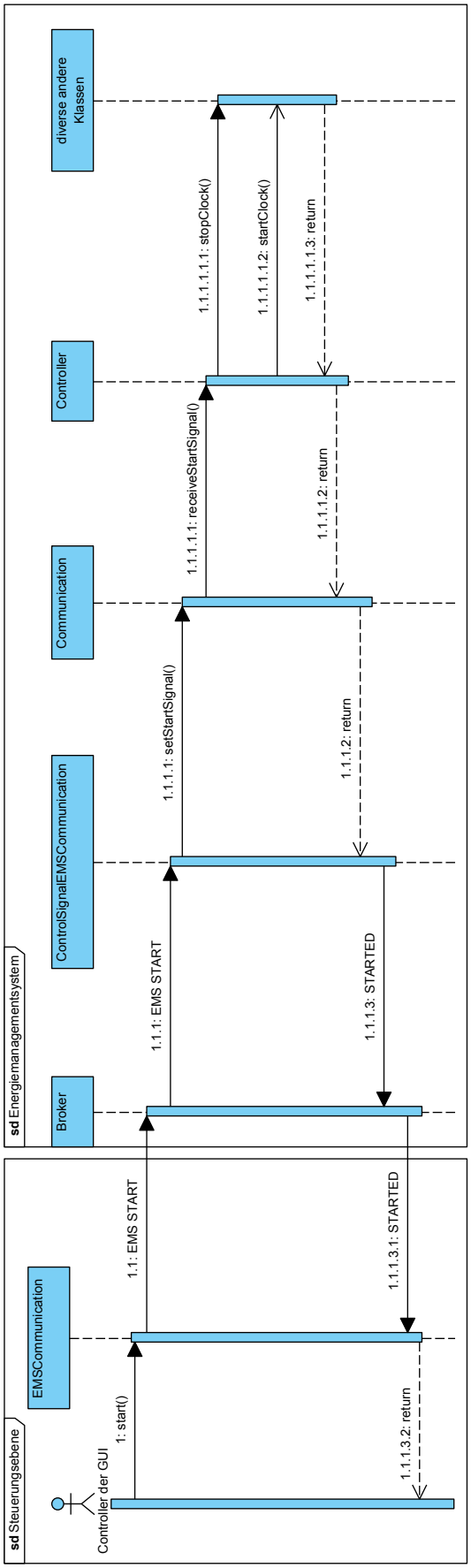


Abb. A.2 Sequenzdiagramm: Senden eines START-Signals von der Steuerungsebene an das EMS (Version 1.0)

A.2 EER-Diagramme

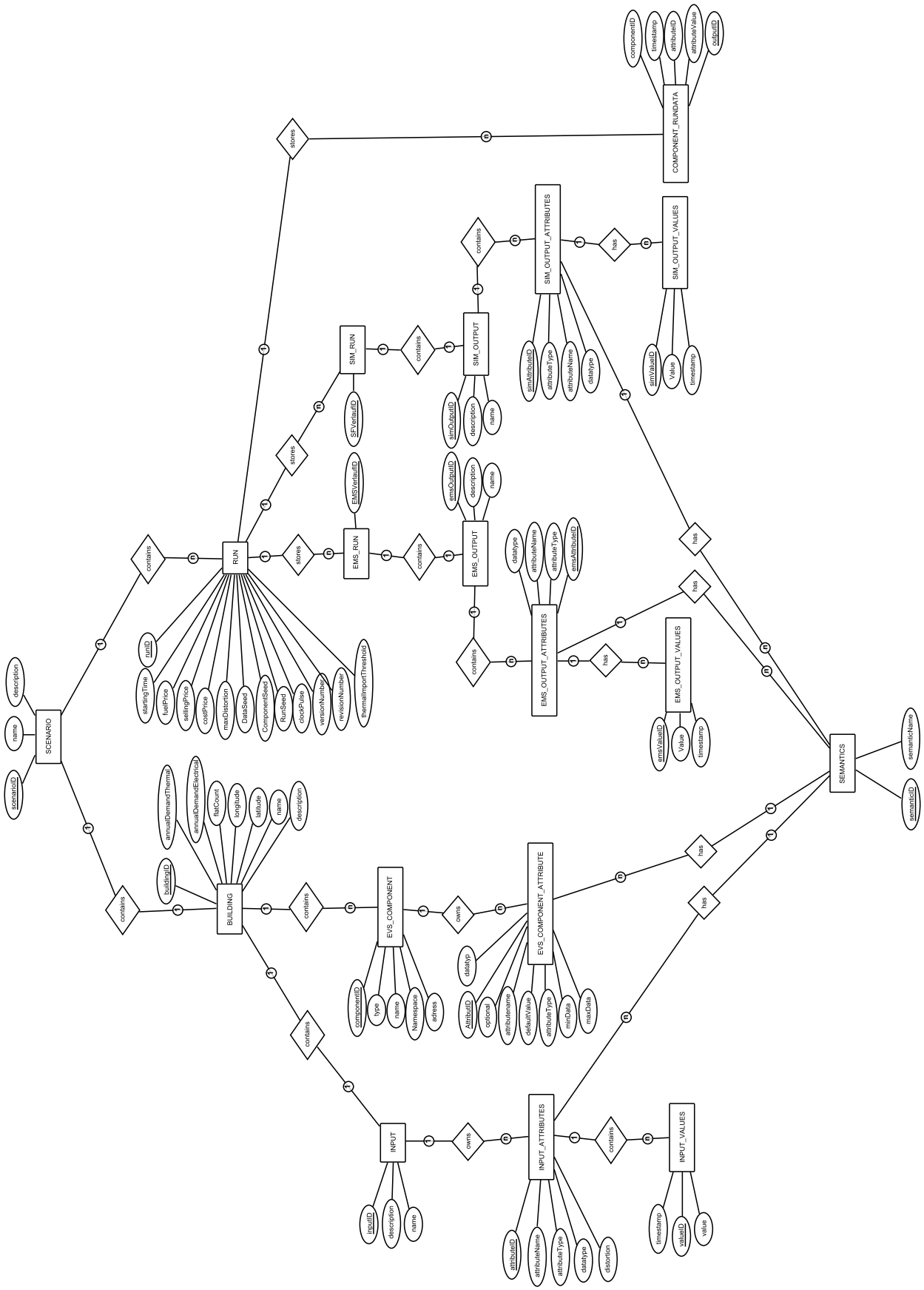


Abb. A.3 EER-Diagramm der Szenario-Datenbank in Baum-Struktur.

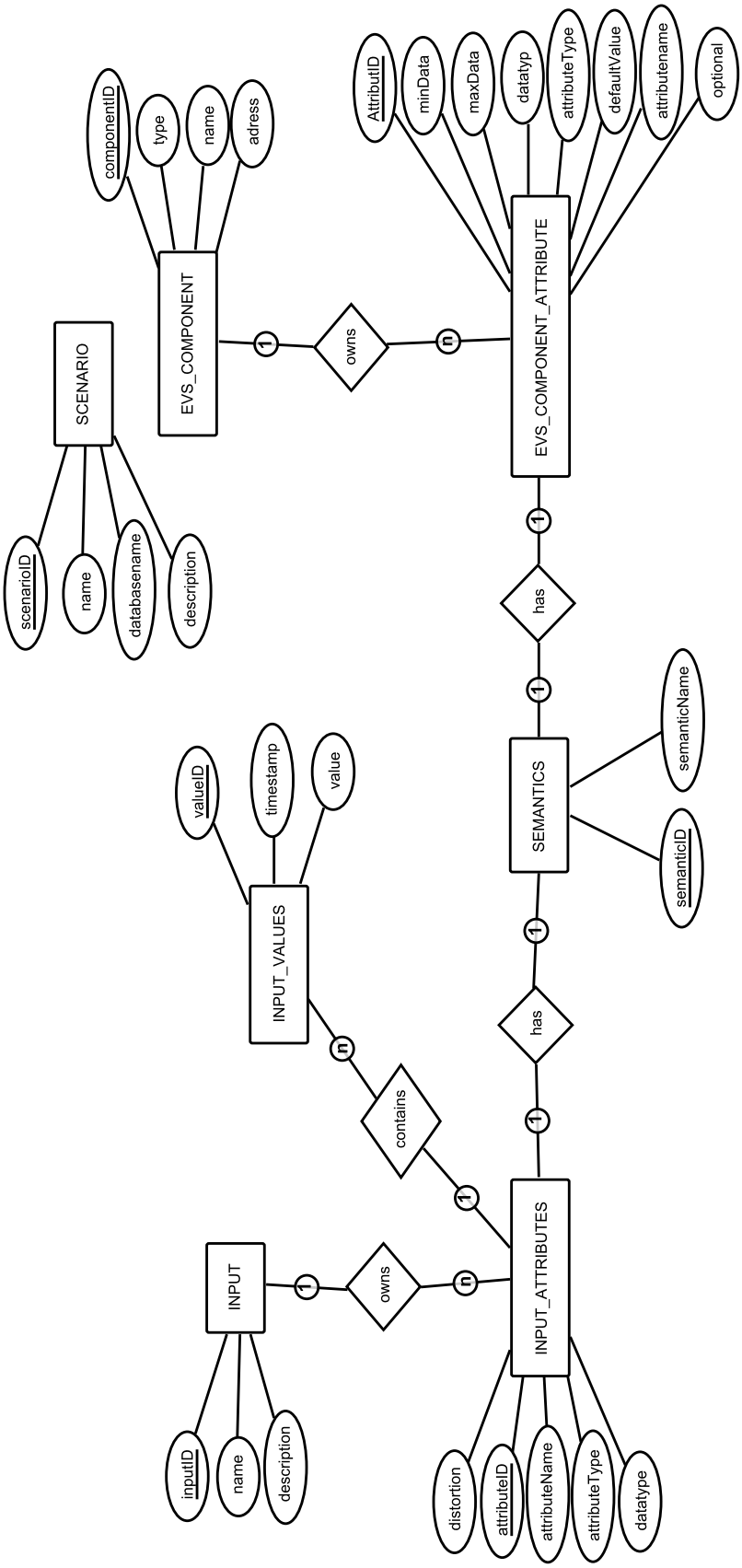


Abb. A.4 EER-Diagramm der Konfigurations-Datenbank.

A.3 Tabellen

Entität	Attribut	MySQL-Datentyp	Beschreibung
BUILDING	buildingID	Integer (Key)	Eindeutiger Identifizierer
	name	Varchar	Der Name, den der Benutzer seiner Konfiguration - dem Gebäude - geben kann
	description	Text	Eine Beschreibung des Gebäudes und seiner Konfiguration
	latitude	Decimal (64, 30)	Die Angabe des Breitengrades zur Bestimmung der Position des Gebäudes
	longitude	Decimal (64, 30)	Die Angabe des Längengrades zur Bestimmung der Position des Gebäudes
	flatCount	Integer	Die Anzahl der Wohneinheiten des Gebäudes
	annualDemandThermal	Decimal (64, 30)	Angabe des jährlichen thermischen Verbrauches
	annualDemandElectrical	Decimal (64, 30)	Angabe des jährlichen elektrischen Verbrauches
RUN	RunSeed	Integer	Seed, mit dem der Run initialisiert wurde
	revisionNumber	Varchar	Die interne Versionsnummer der Software, die etwas genauer ist als das Feld versionNumber, da sie auch einzelne Bugfix-Releases unterscheidet
	versionNumber	Varchar	Der Versionsstand der Software
	runID	Integer (Key)	Eindeutiger Identifizierer

Entität	Attribut	MySQL-Datentyp	Beschreibung
	costPrice	Decimal (64, 30)	Angabe, wie teuer zugekaufte Energie ist
	startingTime	Varchar	Angabe, in welchem Zeitintervall der Lauf startete
	sellingPrice	Decimal (64, 30)	Preis, für den Überschüssige Energie verkauft werden kann
	dataSeed	Integer	Seed, mit dem das Verrauschen der Prognosedaten vorgenommen wird
	componentSeed	Integer	Seed, der für die Initialisierung der Zufallszahlengeneratoren der Komponenten genutzt wird
	clockPulse	Integer	Zeitangabe des Intervalls zwischen den einzelnen Takten
	maxDistortion	Decimal (64, 30)	Angabe in Prozent, wie stark die verrauschten Daten von den Prognosedaten abweichen dürfen
EMS_RUN	emsRunID	Integer (Key)	Eindeutiger Identifizierer
SIM_RUN	simRunID	Integer (Key)	Eindeutiger Identifizierer
COMPONENT_-RUNDATA	componentID	Integer	Eindeutiger Identifizierer der Komponente, von der die Daten stammen
	timestamp	Integer	Zeitstempel, der das Intervall angibt, zu dem die Daten gespeichert wurden
	outputID	Integer (Key)	Eindeutiger Identifizierer
	attributeID	Integer	Eindeutiger Identifizierer des Attributes, zu dem der Wert gehört

Entität	Attribut	MySQL-Datentyp	Beschreibung
	attributeValue	Integer	Wert des gespeicherten Attributes
EVS_-COMPONENT	name	Varchar	Name der EVS-Komponente
	adress	Varchar	IP-Adresse der EVS-Komponente
	type	Varchar	Typ der EVS-Komponente
	componentID	Integer (Key)	Eindeutiger Identifizierer
EVS_-COMPONENT_-ATTRIBUTE	maxData	Decimal (64, 30)	maximaler Wert, den das Attribut annehmen kann
	optional	Boolean	Boolesche Angabe, ob dieser Wert erforderlich oder optional ist
	attributeType	Varchar	Typ des Attributes
	datatyp	Varchar	Datentyp des Attributes
	AttributeID	Integer (Key)	Eindeutiger Identifizierer
	minData	Decimal (64, 30)	minimaler Wert, den das Attribut annehmen kann
	attributename	Varchar	Name des Attributes
	defaultValue	Varchar	Standardwert des Attributes, wenn der Benutzer keine andere Angabe gemacht hat
INPUT_-VALUES	value	Decimal (64, 30)	Wert der Eingabeparameters
	timestamp	Varchar	Zeitstempel, an dem der Eingabewert aktuell war
	valueID	Integer (Key)	Eindeutiger Identifizierer
INPUT	description	Text	Beschreibung des Eingabeparameters
	inputID	Integer (Key)	Eindeutiger Identifizierer

Entität	Attribut	MySQL-Datentyp	Beschreibung
	name	Varchar	Name des Eingabeparameters
INPUT_-ATTRIBUTES	attributeType	Varchar	Typ des Eingabeattributes
	attributeID	Integer (Key)	Eindeutiger Identifizierer
	attributeName	Varchar	Name des Eingabeattributes
	datatype	Varchar	Datentyp des Eingabeattributes
	distortion	Boolean	Angabe, ob der Wert bereits verrauscht wurde
SCENARIO	description	Text	Beschreibung des Scenarios
	scenarioID	Integer (Key)	Eindeutiger Identifizierer
	name	Varchar	Name, den der Benutzer dem Szenario gegeben hat
SEMANTICS	semanticName	Varchar	Name der Semantik
	semanticID	Integer (Key)	Eindeutiger Identifizierer
EMS_-OUTPUT_-VALUES	emsValueID	Integer (Key)	Eindeutiger Identifizierer
	value	Varchar	Wert der Ausgabe
	timestamp	Varchar	Zeitstempel, an der die Ausgabe aktuell war
SIM_-OUTPUT_-VALUES	value	Varchar	Wert der Ausgabe der Simulationskomponente
	simValueID	Integer (Key)	Eindeutiger Identifizierer
	timestamp	Varchar	Zeitstempel an der die Ausgabe aktuell war
EMS_-OUTPUT	emsOutputID	Integer (Key)	Eindeutiger Identifizierer
	description	Text	Beschreibung der Ausgabe des EMS

Entität	Attribut	MySQL-Datentyp	Beschreibung
	name	Varchar	Name der Ausgabe des EMS
SIM_-OUTPUT	name	Varchar	Name der Ausgabe des Simulationsframeworks
	description	Text	Beschreibung der Ausgabe des Simulationsframeworks
	simOutputID	Integer (Key)	Eindeutiger Identifizierer
EMS_-OUTPUT_-ATTRIBUTES	emsAttributeID	Integer (Key)	Eindeutiger Identifizierer
	attributeType	Varchar	Typ der Ausgabe des EMS
	attributeName	Varchar	Name des Ausgabe-Attributes des EMS
	datatype	Varchar	Datentyp des Ausgabe-Attributes des EMS
SIM_-OUTPUT_-ATTRIBUTES	datatype	Varchar	Datentyp des Ausgabe-Attributes des Simulationsframeworks
	attributeType	Varchar	Typ der Ausgabe des Simulationsframeworks
	attributeName	Varchar	Name des Ausgabe-Attributes des Simulationsframeworks
	simAttributeID	Integer (Key)	Eindeutiger Identifizierer

Tabelle A.1 Tabellenbeschreibung der Konfigurations-Datenbank

Tabelle		**Beschreibung**
Attribut	**Datentyp**	**Beschreibung**
Scenario		Alle Szenarien sind auswählbar über die ConfigDB. Das Zusammengestellte Szenario ist in der ScenarioDB enthalten.
ScenarioID	int	Eindeutiger Identifizierer
Name	varchar	Der Name des Szenarios
Description	text	Eine kurze Beschreibung des Szenarios
databasename	varchar	Der Datenbankname des Szenarios mit allen Laufdaten
EVS_Component		Basis-Daten einer EVS-Komponente
ComponentID	int	Eindeutiger Identifizierer
Type	varchar	Typ der Komponente: PV-Anlage, Kühlhaus, Batterie...
Name	varchar	Name der Komponente
Namespace	varchar	Namespace der Komponente auf dem Koppler
Adress	varchar	Adresse der Komponente auf dem Koppler
EVS_Component_Attribute		Attribut einer EVS-Komponente Bspw. Sun-Azimuth für eine PV-Anlage
AttributeID	int	Eindeutiger Identifizierer
minData	decimal	Typ der Komponente: PV-Anlage, Kühlhaus, Batterie...
maxData	decimal	Name der Komponente
datatyp	varchar	Namespace der Komponente auf dem Koppler
attributeType	varchar	SBIN, SIIN, EBIN, ... (siehe Tabelle A.2)
defaultValue	varchar	Voreinstellung für den Attributwert
attributeName	varchar	Name des Attributs
optional	tinyint	Gibt an ob das Attribute angegeben werden muss.
semanticID	int	Eindeutiger Fremdschlüssel zur Semantik
componentID	int	Eindeutiger Fremdschlüssel zur Komponente
Input		Input-Daten sind von außen in die Simulation gegebene Daten. Momentan sind dies Wetterdaten und Grundlastdaten.
inputID	int	Eindeutiger Identifizierer
name	varchar	Name des Inputs
description	text	Kurze Beschreibung des Inputs
Input_Attribute		Attribut der Input-Daten. Bspw. Temperatur bei Wetterdaten
attributeID	int	Eindeutiger Identifizierer
attributeName	varchar	Name des Attributs
attributeType	varchar	Kurzer Beschreibungstext des Input-Attributs
datatype	varchar	forecast, baseload, ...
distortion	tinyint	Verrauschungs-Flag: 0 = Dieser Wert soll nicht verrauscht werden 1 = Dieser Wert soll verrauscht werden
semanticID	int	Eindeutiger Fremdschlüssel zur Semantik
inputID	int	Eindeutiger Fremdschlüssel zum Input
Input_Value		Werte der Input-Attribute zu bestimmten Zeitpunkten
valueID	int	Eindeutiger Identifizierer
value	decimal	Der Wert des Attributes zu einem bestimmten Zeitpunkt
timestamp	varchar	Der Zeitpunkt des Eingabewertes
attributeID	int	Eindeutiger Fremdschlüssel zum Attribut
Semantic		Semantiken ermöglichen das Mapping von Input und Output-Daten auf zugehörige Parameter der EVS-Komponenten
semanticID	int	Eindeutiger Identifizierer
semanticName	varchar	Der Name der Semantik. Dieser Name ist einmalig zu vergeben. Der Name kann von mehreren Attributen verwendet werden.

Tabelle A.2 Beschreibung der Attribut-Typen von EVS-Komponenten

AttributeType	Beschreibung
EBOT	EMS Output „EBOT“: EBOT kennzeichnet Outputs der Komponente, die vom EMS gelesen werden können.
EBIN	EMS Sampleindex „EBIN“: EBIN kennzeichnet den Input der Komponente, mit dem das EMS nach jedem Optimierungsschritt das gewählte Sample der EVS-Komponente mitteilt.
ESIN	EMS Sampling input „ESIN“: ESIN kennzeichnet ein Input der Komponente, die vom EMS für das Sampling gesetzt werden müssen.
ESOT	EMS Sampling output „ESOT“: ESOT kennzeichnet Outputs der Komponente, die vom EMS während des Samplings gelesen werden können.
EBFL	EMS-Flag „EBFL“: Jede Komponente besitzt ein EBFL. Dieses Flag gibt an, dass das Energiemanagementsystem seine Werte in die Inputs der Komponente geschrieben hat.
SIIN	SF Config Parameter „SIIN“: SIIN kennzeichnet Inputs einer Komponente, die für die Konfiguration beim Initialisieren vom SF gesetzt werden müssen.
SBIN	SF Input „SBIN“: SBIN kennzeichnet Inputs, die vor dem Optimierungsschritt vom Simulationsframework für das Sampling der Komponente gesetzt werden müssen.
SBFL	SF-Flag „SBFL“: Jede Komponente besitzt ein SBFL. Dieses Flag gibt an, dass das Simulationsframework die Inputs der Komponente fertig geschrieben hat.
SIFL	SF-Init-Falg „SIFL“: Jede Komponente besitzt ein SIFL. Dieses Flag gibt an, das alle für die Konfiguration notwendigen Input-Werte beim Initialisieren gesetzt wurden.
ESFL	Sampling-Flag „ESFL“: Jede Komponente mit Sampling hat ein ESFL. Dieses gibt an, das mit dem Sampling begonnen werden soll (ESFL = 1).

CARL VON OSSIETZKY universität OLDENBURG

Department für Informatik – Abteilung Umweltinformatik

Seminarband

Sommersemester 2014

Vorwort

Im Rahmen der Projektgruppe *Hardwarebasierte Simulation energieautonomer Gebäude* erfolgte eine Seminarphase im Sommersemester 2014 an der Carl von Ossietzky Universität Oldenburg, Department für Informatik, Abteilung Umweltinformatik.

Dieses Buch enthält die im Zuge der Seminarphase entstandenen Einzelbeiträge.

Oldenburg,
Juli 2014

Inhaltsverzeichnis

Referenten

Prof. Dr. Michael Sonnenschein (Editor)
sonnenschein@informatik.uni-oldenburg.de

Dr. Ute Vogel (Editor)
vogel@informatik.uni-oldenburg.de

M.Sc. Christian Hinrichs (Editor)
hinrichs@informatik.uni-oldenburg.de

Tobias Kölker
tobias.koelker@uni-oldenburg.de

Marius Hacker
marius.hacker@uni-oldenburg.de

Michael Cordes
michael.cordes@uni-oldenburg.de

Dag Ennenga
dag.ennenga@uni-oldenburg.de

Eugen Langolf
eugen.langolf@uni-oldenburg.de

Connor Fibich
connor.fibich@uni-oldenburg.de

Kevin Möhlmann
kevin.moehlmann@uni-oldenburg.de

Hendrik Grunau
hendrik.grunau@uni-oldenburg.de

Sönke Martens
soenke.martens@uni-oldenburg.de

Claas Martin
claas.martin@uni-oldenburg.de

Christian Best
christian.best@uni-oldenburg.de

Kapitel 1
Mini-Blockheizkraftwerke und Wärmepumpen

Tobias Koelker

Zusammenfassung Die vorliegende Arbeit ist eine Einführung in die Technologien Mini-Blockheizkraftwerk und Wärmepumpe, die im Zuge der Energiewende zur Realisierung einer nachhaltigeren Energiegewinnung eine bedeutende Rolle spielen können. Dabei werden einerseits die Grundlagen der Technologien erläutert, andererseits die Verknüpfung zum Thema der Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“ hergestellt.

1.1 Motivation

Das weltweite Vorkommen der fossilen Brennstoffe, also der verfügbaren und zugänglichen Primärenergie wie beispielsweise Erdöl und Erdgas, ist limitiert und schon in naher Zukunft wird die Menschheit an die Grenzen der Ressourcennutzung stoßen und vor der Frage stehen, wie die Erde weiter mit Energie versorgt werden kann. Das dramatischste Beispiel ist der fossile Rohstoff Erdöl.

Erdöl ist der einzige nicht erneuerbare Energierohstoff, bei dem in den kommenden Jahrzehnten eine steigende Nachfrage wahrscheinlich nicht mehr gedeckt werden kann.[1]

In Europa ist der verhältnismäßig geringfügig vorkommende Rohstoff schon zur Hälfte aufgebraucht und auf fast allen anderen Kontinenten ist mindestens ein Viertel des Rohöls bereits gefördert worden. Da zudem noch die Förderung jährlich ansteigt und angesichts der Tatsache, dass die Umstellung auf alternative Energiesysteme lange Zeiträume benötigt, ist es erforderlich, frühzeitig zu agieren anstatt auf eine Rohstoffknappheit zu reagieren. Ein vollständiges Ausschöpfen des Rohstoffs Öl oder Gas würde zum jetzigen Zeitpunkt katastrophale Auswirkungen nicht nur in Hinblick auf Wirtschaft und Politik, sondern auf das gesamte Weltgefüge haben.

Carl von Ossietzky Universität Oldenburg
E-mail: tobias.koelker@uni-oldenburg.de

Um den Aspekt Nachhaltigkeit zu berücksichtigen, muss mit der Nutzung der limitierten Ressourcen sorgsam umgegangen werden. Die Entwicklung und Einführung der Nutzung von erneuerbaren Energien, hauptsächlich die Wind- und Solarenergie, leistet zwar ihren Beitrag zur Energiegewinnung, jedoch können diese Technologien nicht die vollständige Energieversorgung sicherstellen. Die primären Energieträger sind und werden auch noch in Zukunft die Hauptenergieträger bleiben. Um jedoch auch für die folgenden Generationen zu gewährleisten, dass fossile Rohstoffe zur Verfügung stehen, gibt es zwei Möglichkeiten. Entweder wird der weltweite Energieverbrauch gesenkt, was jedoch aufgrund des immerzu angestrebten Wirtschaftswachstum, welches Energie benötigt, und des steigenden Konsumverhaltens der Menschen insbesondere der westlichen Welt eigentlich kaum zu realisieren ist, oder die Effizienz der Energiegewinnung muss gesteigert werden. Wenn man die Wirkungsgrade der konventionellen Kraftwerke, zu denen in erster Linie Kohle-, Gas- und Kernkraftwerke zählen, die in Deutschland rund drei Viertel der Stromerzeugung ausmachen, betrachtet, kommt man zu dem Schluss das Letzteres die Lösung des Problems Energieversorgung sein muss. Doch wie kann man die Effizienz der Energiegewinnung erhöhen? Die Antwort heißt Kraft-Wärme-Kopplung, kurz KWK. Durch die KWK kann der Wirkungsgrad bezüglich eingesetzter Primärenergie auf bis zu 90 Prozent erhöht werden und so der Verbrauch dieser erheblich reduziert werden. In der öffentlichen Diskussion wird die KWK bereits als zentraler Baustein zum Einen für das Erreichen der Energieeffizienzziele, zum Anderen zur CO2-Reduktion angesehen. Auf politischer Ebene gibt es seit 1. April 2002 das Kraft-Wärme-Kopplungsgesetz, welches bereits zwei größere Novellierungen hinter sich hat.

> „Zweck des Gesetzes ist es, im Interesse der Energieeinsparung, des Umweltschutzes und der Erreichung der Klimaschutzziele der Bundesregierung einen Beitrag zur Erhöhung der Stromerzeugung aus Kraft-Wärme-Kopplung in der Bundesrepublik Deutschland auf 25 Prozent bis zum Jahr 2020 durch die Förderung der Modernisierung und des Neubaus von Kraft-Wärme-Kopplungsanlagen (KWK-Anlagen), die Unterstützung der Markteinführung der Brennstoffzelle und die Förderung des Neu- und Ausbaus von Wärme- und Kältenetzen sowie des Neu- und Ausbaus von Wärme- und Kältespeichern, in die Wärme oder Kälte aus KWK-Anlagen eingespeist wird, zu leisten.“ (Nach der neuen Fassung vom 12. Juli 2012)

Vor diesem Hintergrund wird in der folgenden Ausarbeitung das Thema KWK und die damit zusammenhängenden Technologie Blockheizkraftwerk behandelt. Des Weiteren wird die Wärmepumpe vorgestellt, die mit hohen Wirkungsgrad Heizwärme für Gebäude bereitstellen kann. Begrenzt wird die Ausarbeitung auf die durch die Projektgruppe (Hardwarebasierte Simulation energieautonomer Gebäude) gegebenen Rahmenbedingung.

1.2 Grundlagen der Technologien

Das folgende Kapitel thematisiert die Technologien Mini-Blockheizkraftwerk und Wärmepumpe. Dabei werden zum Einen Aufbau und Funktionsweise der Technolo-

gien beschrieben, zum Anderen verschiedene Betriebsarten und Steuerungsmöglichkeiten vorgestellt.

1.2.1 Mini-Blockheizkraftwerk

Blockheizkraftwerke, kurz BHKW, sind Anlagen, die auf Basis von Verbrennungsmotoren oder Gasturbinen arbeiten. Sie nutzen im Gegensatz zu konventionellen Kraftwerken, die in Deutschland den Großteil des Strombedarfs decken, die bei der Erzeugung von elektrischer Energie entstehende thermische Energie als Heizwärme. Das Prinzip, welches dahinter steckt, ist die Kraft-Wärme-Kopplung. Kleine BHKW, die sogenannten Mini-BHKW, befinden sich direkt am Ort des Strom- und Wärmeverbrauchs und können die resultierende Wärmeenergie aufgrund der sehr geringen Entfernung effektiv nutzen. Durch die Nutzung der Kraft-Wärme-Kopplung können sehr hohe Wirkungsgrade von bis zu über 90 Prozent erreicht werden. Im Vergleich dazu erzielen konventionelle Kraftwerke in der Regel Wirkungsgrade von maximal 60 Prozent.

Das Spektrum von BHKW allgemein reicht bis hin zu Groß-Anlagen mit einer elektrischen Leistung von 10MW. Mini-BHKW, die in ihren Einsatz in Häusern, Wohngebäuden oder kleinen Industrieanlagen finden, haben meist eine elektrische Leistung von bis zu 15kW.

1.2.1.1 Aufbau

Abbildung 1.1 zeigt den Aufbau eines Mini-BHKW. Ein Mini-BHKW besteht in erster Linie aus einem Block, der namensgebend für diese Art von Anlagen ist. Dieser Block besteht aus einer Kraftmaschine, einem Verbrennungsmotor beziehungsweise einer Gasturbine, und einem Generator. In der Kraftmaschine wird die Energie aus dem Brennstoff in mechanische Energie umgewandelt, die wiederum im Generator in elektrische Energie umgesetzt wird. Diese Energie kann direkt nutzbar gemacht werden. Die Abwärme der Stromerzeugung entsteht hauptsächlich in der Kraftmaschine. Diese Wärme erhitzt Wasser, welches in den Heizkreis des Gebäudes mit eingekoppelt und als Nutzwärme verwendet wird. Der Rücklauf der Heizung ist abgekühltes Wasser, das zur Kühlung die Kraftmaschine durchströmt und dabei wieder erhitzt wird. Der Kreislauf beginnt von vorne. Eine weitere Komponente des BHKW kann ein Heizkessel bilden. Dieser dient als Reserve- beziehungsweise Pufferspeicher für das Heizsystem. Der Kessel wird durch die entstehende Wärme erhitzt, falls zu diesem Zeitpunkt keine Heizwärme benötigt wird. Im Falle einer Unterversorgung von Wärmeenergie kann auf die Reserve Heizkessel zurückgegriffen werden.

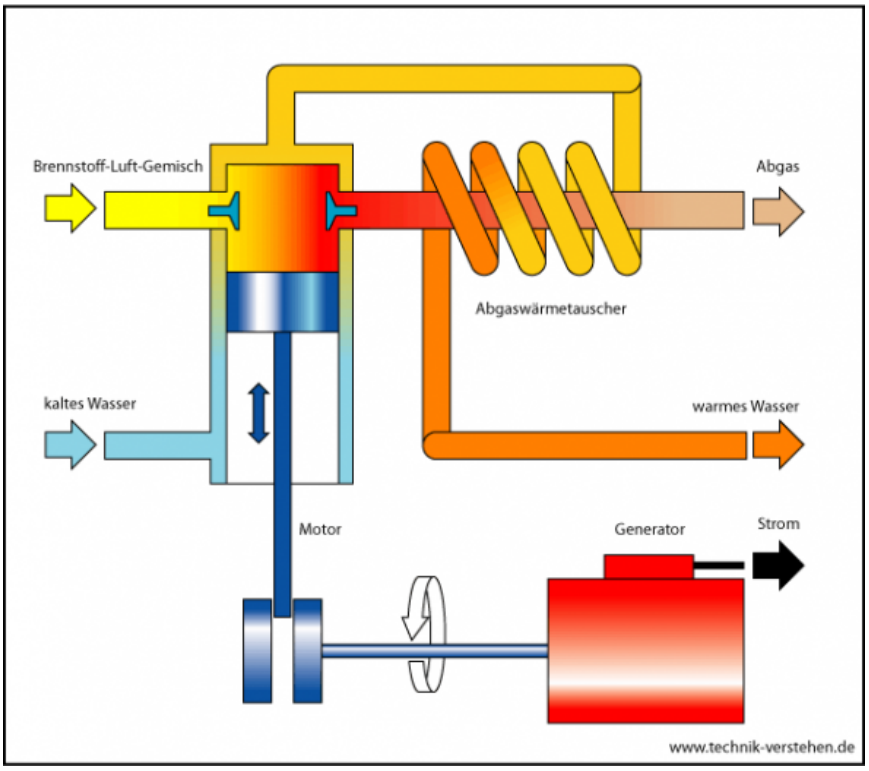


Abb. 1.1 Funktionsschema eines BHKW

1.2.1.2 Physikalische Grundlagen

Zur Beschreibung von Leistung und Effizienz von Blockheizkraftwerken werden verschiedene Kenngrößen verwendet. Die ausgehende Leistung wird durch die elektrische und thermische Leistung beschrieben. Die eingehende Leistung durch die so genannte Brennstoffleistung. Daraus resultiert der Wirkungsgrad des BHKW.

Die elektrische Leistung P_{el} des BHKW ist die Kenngröße für die elektrische Energie, die pro Zeiteinheit von dem BHKW produziert wird. Da die direkt am Generator anfallende Leistung in der Regel größer ist als die tatsächlich eingespeiste Energie, spricht man hierbei von der elektrischen Nettoleistung. Verluste können durch beispielsweise Umwälzpumpen, Gebläse oder eine geräteeigenen Steuerung entstehen. Die Angabe der elektrischen Leistung erfolgt in der Einheit Watt.

Die thermische Leistung $\dot{Q}_{th}$ beschreibt, wie viel Nutzwärme durch das BHKW abgegeben wird. Aus Sicht des Verbrauchers wird diese Leistung auch Wärme- oder Heizleistung genannt, da mit Hilfe der aus der Stromerzeugung resultierenden Wärme ein Zwischenmedium, üblicherweise Wasser, erhitzt und dieses für die Erwär-

mung von Brauchwasser oder der Heizung verwendet wird. Dahingegen dient die Wärmeabfuhr dem BHKW selbst als Kühlung. Um die thermische Leistung des BHKW bestimmen zu können, müssen die Vorlauftemperatur, also die Temperatur des Kreislaufwassers vor dem BHKW, die Rücklauftemperatur und der Wasservolumenstrom gemessen werden. Zur Beschreibung dieser Kenngröße dient ebenfalls die Einheit Watt.

Die Brennstoffleistung $\dot{Q}_{Brennstoff}$ ist die Leistung, die dem BHKW mittels Energieträger zugeführt wird. Sie ist abhängig vom Heizwert des jeweiligen Brennstoffs und wird wie elektrische und thermische Leistung in Watt angegeben.

Aus den drei beschrieben Größen lassen sich die Wirkungsgrade des BHKW berechnen. Als elektrischen Wirkungsgrad bezeichnet man das Verhältnis der elektrischen Leistung zur Brennstoffleistung. Der thermische Wirkungsgrad ergibt sich aus dem Verhältnis der thermischen Leistung zur Brennstoffleistung. Zur allgemeinen Bewertung von BHKW wird der Gesamtwirkungsgrad η_{ges} berechnet.

$$\eta_{ges} = \frac{P_{el} + \dot{Q}_{th}}{\dot{Q}_{Brennstoff}}$$

Dieser ergibt sich aus dem Verhältnis von der Summe aus elektrischer und thermischer Leistung zur Brennstoffleistung. Modernste BHKW können können Wirkungsgrade von über 0,9, also über 90 Prozent, erreichen. Als weitere Kenngröße des BHKW wird die Stromkennzahl berechnet. Sie gibt das Verhältnis von elektrischer Energie zur thermischen Energie wieder. Diese liegt in der Regel bei 0,4 bis 0,5.

1.2.1.3 Betriebsarten

Es gibt drei verschieden Möglichkeiten, je nach Anforderungen an Strom- und Wärmeerzeugung, ein Blockheizkraftwerk zu betreiben.

Eine Option ist das stromgeführte BHKW. Das BHKW wird entsprechend des Bedarfs an elektrischer Leistung ausgelegt und betrieben. Besteht bei dieser Betriebsart ein Mangel an thermischer Energie wird das BHKW durch den Heizkessel beziehungsweise Wärmespeicher unterstützt. Entsteht ein Überschuss an thermischer Energie, reduziert sich der Wirkungsgrad des BHKW, da diese nicht gebrauchte Energie an die Umwelt abgegeben wird.

Beim wärmegeführten BHKW erfolgt die Auslegung und der Betrieb nach der benötigten thermischen Energie des Gebäudes, um Soll-Temperatur zu halten und die Versorgung mit Warmwasser zu gewährleisten. Liegt der Bedarf an elektrischer Energie über den erzeugten Wert, wird die Differenz entweder durch Entnahme von Strom aus dem öffentlichen Netz oder durch eine zusätzliche Energiequelle ausgeglichen.

Des Weiteren ist es möglich, die beiden Betriebsarten Strom- und Wärmeführung zu kombinieren um die Nutzung des BHKW zu optimieren. Dazu wird jedoch eine geeignete Regelungs- und Steuerungstechnik benötigt.

1.2.1.4 Steuerungsmöglichkeiten

Aufgrund der geringen Stromkennzahlen, also dem geringen Verhältnis von elektrischer zur thermischen Energieerzeugung, wird das Blockheizkraftwerk in der Regel zur Wärmeversorgung von Gebäuden eingesetzt. Für das zu beheizende Gebäude ist ein Temperaturbereich definiert. Das BHKW ist dafür verantwortlich, dass die minimale Temperaturgrenzen nicht unterschritten und die Grenze für das Maximum nicht überschritten werden. Die dabei erzeugte elektrische Energie ist ein positiver Nebeneffekt und kann entweder selbst verbraucht oder gegen eine Vergütung in das Stromnetz eingespeist werden. Bezüglich dieser Anforderungen bietet das BHKW verschiedene Freiheitsgrade, die eine Optimierung der Nutzung ermöglichen. Zum Einen können die Aktivierungszeitpunkte des BHKW zeitlich verschoben werden, zum Anderen bietet die Regulierbarkeit der Auslastung zusätzlichen Spielraum. In Abbildung 1.2 sind zwei Möglichkeiten für die Steuerung eines BHKW dargestellt.

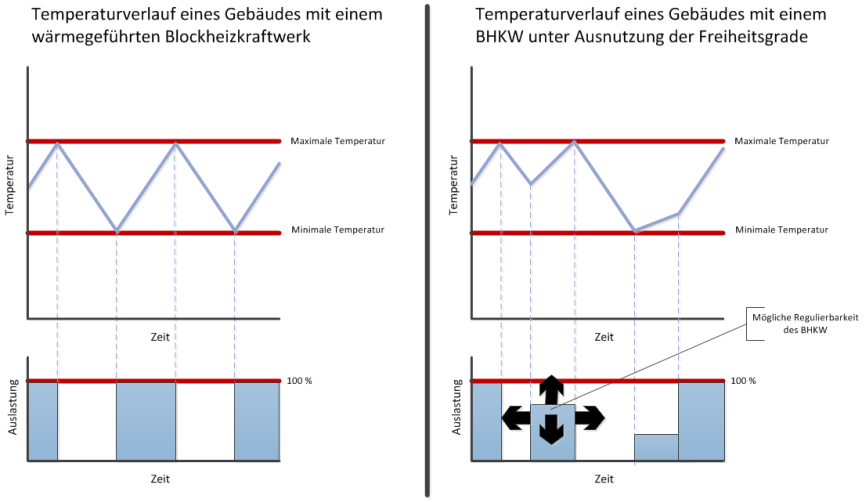


Abb. 1.2 Beispiele für die Steuerung eines BHKW

Abgebildet sind die Temperaturverläufe eines Gebäudes in Abhängigkeit von der Auslastung des BHKW. Der linke Teil der Abbildung zeigt die einfachste Art der Steuerung. Das BHKW läuft unter Vollast bis die maximale Temperaturgrenze erreicht ist und schaltet sich dann ab bis die minimale Temperaturgrenze unterschritten wird. Diese einfache Art des Betriebs ist zwar ausreichend zum Einhalten der Temperaturgrenzen, jedoch nicht sehr effizient, da keine Berücksichtigung der erzeugten und benötigten elektrische Energie oder anderer Energieerzeuger stattfindet.

Erhöht man die Komplexität des Systems und will die Effizienz dabei maximieren, ist eine aufwendigere Steuerung unter Berücksichtigung des Strombedarfs und

möglicher zusätzlicher Energieerzeuger wie Wärmepumpe oder Photovoltaikanlage notwendig. Hierbei bietet das BHKW den Vorteil, dass eine flexible Steuerung unter Verwendung der Freiheitsgrade möglich ist. Im rechten Teil der Abbildung 1.2 ist Beispielhaft ein Temperaturverlauf eines Gebäudes unter Verwendung der gebotenen Freiheitsgrade dargestellt. Das BHKW wird schon zu einem früheren Zeitpunkt, bevor die minimale Temperaturgrenze erreicht ist, wieder eingeschaltet um beispielsweise aktuelle Lastspitzen abzudecken. Durch eine verringerte Leistungsaufnahme des BHKW und der daraus resultierenden geringeren Erzeugung von thermischer und elektrischer Energie lässt sich der Winkel der Kurve des Temperaturverlaufs ändern. Einen zusätzlichen Freiheitsgrad kann man durch den Einsatz eines Pufferspeichers gewinnen. Ist beispielsweise das Temperaturminimum erreicht, kann dieser zum Einsatz kommen um das BHKW für einen längeren Zeitraum auszuschalten.

Zusammenfassend ist das BHKW ein Energieerzeuger, der eine gute Komponente für die Realisierung von Gebäuden, die durch eine möglichst hohen Autarkiegrad bezüglich der Strom- und Wärmeversorgung gekennzeichnet sind, darstellt. Gerade im Zusammenspiel mit anderen Energieerzeugern sorgen die gebotenen Freiheitsgrad für viel Spielraum hinsichtlich der Steuerung und Regulierung.

1.2.2 Wärmepumpe

Wärmepumpen sind Anlagen, die mit Hilfe elektrischer Antriebsleistung thermische Energie aus der Umgebung auf ein für Heizzwecke verwendbares Temperaturniveau anheben. Als mögliche umgebende Wärmequellen dienen Luft, Grundwasser und das Erdreich. Wärmepumpen nutzen nicht das Prinzip der Kraft-Wärme-Kopplung, sondern dienen lediglich der Erzeugung von Heizwärme für Gebäude. Sie zeichnen sich durch eine sehr Effizienz aus, sodass sie bei optimaler Verwendung in der Lage sind, den thermischen Energiebedarf eines Gebäudes abhängig von der Betriebsart vollständig oder größtenteils zu decken.

1.2.2.1 Aufbau und Funktionsweise

Eines aus der Thermodynamik bekannte Phänomen ist die Erwärmung von Gasen bei der Kompression und deren Abkühlung bei der Expansion. Dieses Prinzip nutzt die Wärmepumpe um Heizwärme zu gewinnen.

Abbildung 1.3 zeigt den geschlossenen Kreislauf der Wärmepumpe, in dem ein Wärme- beziehungsweise Arbeitsmittel, beispielsweise Propan oder ein anderer reiner Kohlenwasserstoff, zirkuliert. Dieses durchläuft vier Phasen, dem Verdampfen und Verflüssigen sowie der Kompression und Expansion. Am Anfang des Kreislaufs befindet sich das flüssige Arbeitsmittel im Verdampfer. Hier nimmt das Arbeitsmittel die Wärme aus der Umwelt auf, deren Temperatur stets höher als der Siedepunkt des Arbeitsmittels sein muss. Das Arbeitsmittel verdampft und gelangt zum Verdichter, die zentrale Komponente der Wärmepumpe, welche im Wesentlichen dafür

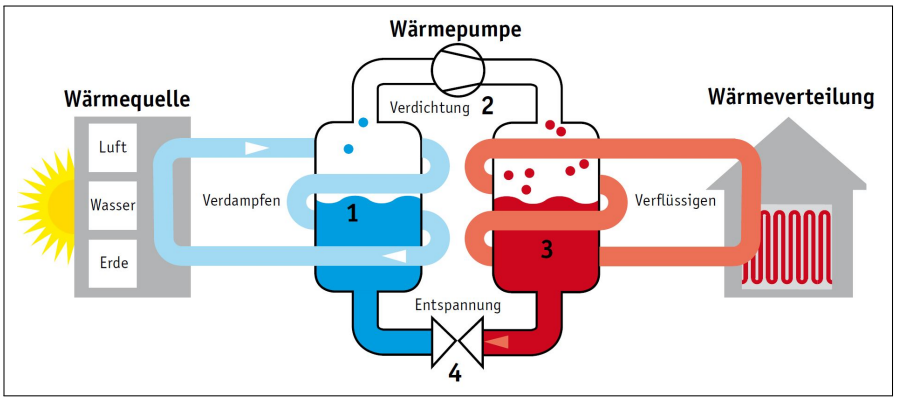


Abb. 1.3 Funktionsschema Wärmepumpe

verantwortlich ist, ob es sich um eine gute oder schlechte Wärmepumpe handelt. Im Verdichter wird das Arbeitsmittel durch den Einsatz von elektrischer Energie komprimiert. Durch diesen Prozess steigt der Druck und die Temperatur des dampfförmigen Arbeitsmittels. Im nächsten Schritt gelangt der Dampf in den Verflüssiger, welcher vom Heizwasser mit niedrigerem Temperaturniveau umgeben ist. Der Dampf kühlt ab und das Arbeitsmedium wird wieder flüssig. Die Wärme wird auf das Wasser übertragen, welches dann als Heizwärme nutzbar ist. Das flüssige Arbeitsmittel gelangt mittels Expansionsventil, in dem sich das Arbeitsmittel wieder entspannt, zurück zum Verdampfer.

Die Umweltwärme kann mit verschiedenen Verfahren gewonnen werden. Zum Einen gibt es die Möglichkeit, die den Verdampfer umgebende Luft zu nutzen. Dieses Verfahren ist jedoch nicht bewährt und spielt daher nur eine untergeordnete Rolle. Zum Anderen gibt es die effizientere Möglichkeit, das Grundwasser oder das Erdreich, in dem das Temperaturniveau schon in zehn Metern Tiefe ganzjährig circa zehn Grad beträgt, als Wärmequelle zu nutzen. Eine Umgebungstemperatur von zehn Grad ist bereits ausreichend um dem entsprechenden Arbeitsmittel genügend Wärme zur Verdampfung hinzuzufügen. Um das Erdreich als Quelle zu nutzen gibt es entweder die Option, die Wärme mittels Erdwärmesonde, welche je nach Beschaffenheit in bis zu 100 Meter Tiefe verlegt wird, der Umwelt zu entziehen oder die Variante des Erdwärmekollektors. Hier wird in etwa einem Meter Tiefe ein horizontales Rohrleitungssystem verlegt, welches die Umweltwärme an den Verdampfer leitet. Die Wahl der Wärmequelle ist in erster Linie von den Gegebenheiten am Ort der Wärmegewinnung abhängig.

Die Effizienz von Wärmepumpen wird mit der Leistungszahl e beschrieben. Sie resultiert aus dem Verhältnis von nutzbarer Wärme und der elektrischen Antriebsleistung. Es können Leistungszahlen von bis zu fünf erreicht werden. Das bedeutet, dass aus 1kW elektrische Leistung bis zu 5kW thermische Energie gewonnen werden können, was eine sehr hohe Effizienz kennzeichnet.

1.2.2.2 Betriebsarten

Auch bei der Wärmepumpe gibt es mehrere mögliche Betriebsarten. Bei der monovalenten Wärmepumpe wird der ganzjährige Wärmebedarf vollständig durch die Wärmepumpe gedeckt. Die Dimensionierung orientiert sich an der niedrigsten Außentemperatur, die im Jahr erreicht wird, sodass auch für die kältesten Tage die Wärmeversorgung gewährleistet wird. Hierbei handelt es sich jedoch um eine unübliche und ineffiziente Betriebsart, da das Potential der Wärmepumpe über den Großteil der Zeit nicht ausgeschöpft wird. Deswegen gibt es andere Betriebsarten, bei denen die Wärmepumpen niedriger dimensioniert und an Tagen mit hohem Heizwärmebedarf durch zusätzliche Heizungen unterstützt werden.

1.2.2.3 Steuerungsmöglichkeiten

Die standardmäßige Wärmepumpe bietet hinsichtlich der Steuerbarkeit einen Freiheitsgrad weniger als das Blockheizkraftwerk, da sie in der Leistungsaufnahme nicht wie das BHKW regulierbar ist. Sie verfügt lediglich über die Zustände Ein und Aus, sodass man keinen Einfluss auf den Steigungswinkel (vgl. Abbildung 1.2) der Temperaturkurve nehmen kann. Dieser ist alleine abhängig von der thermischen Leistung der Wärmepumpe.

Eine Erweiterung im Hinblick auf die Steuerung zu dieser Variante bieten Wärmepumpen im sogenannten 2-Verdichter Betrieb. Diese Anlage bestehen aus zwei Verdichtern, die unabhängig voneinander geregelt werden können. Daraus resultieren drei verschiedene Modulationsstufen:

- Beide Verdichter sind zugeschaltet. Wärmepumpe läuft mit voller Leistung.
- Nur einer der Verdichter ist zugeschaltet.
- Beide Verdichter sind abgeschaltet.

1.3 Fazit

Vergleicht man alleine die Gesamtwirkungsgrade konventioneller Kraftwerke mit denen von Blockheizkraftwerk und Wärmepumpe, lässt sich schlussfolgern, dass die hier vorgestellten Technologien im Bezug auf die Effizienz den konventionellen Kraftwerken deutlich überlegen sind. Doch hierbei ist zwischen der zentralen und dezentralen Energieerzeugung zu unterscheiden. Die konventionelle Kraftwerke, am bedeutendsten sind Kohle-, Gas- und Kernkraftwerke, erzielen höchstens Gesamtwirkungsgrade von 60 Prozent, da die Wärme, die bei Stromerzeugung entsteht, in der Regel nicht weiter genutzt, sondern an die Umgebung abgeben wird. Im Hinblick auf die limitierten Vorkommen fossiler Energieträger auf unserem Planeten muss die Entwicklung in naher Zukunft in Richtung der dezentralen Energieerzeugung führen, um zumindest deren zeitliche Verfügbarkeit zu verlängern und die mit deren Umsetzung verbundene Belastung der Umwelt in Grenzen zu halten. Bei dieser

Entwicklung können die vorgestellten Technologien BHKW und Wärmepumpe eine bedeutende Rolle spielen. Die geringer dimensionierten BHKW nutzen das Prinzip der Kraft-Wärme-Kopplung und erzielen auf diese Weise signifikant höhere Wirkungsgrade. Die Wärmepumpe ist mit einer sehr hohen Leitungszahl in der Theorie ebenfalls eine sehr effiziente Technologie. Da sie ausschließlich thermische Energie erzeugt, ist sie ausschließlich für die Erzeugung von Heizwärme oder zur Erhitzung von Heizkesseln, die als Pufferspeicher dienen, einsetzbar.

Langfristig gesehen sind aber auch diese Technologien keine Lösung für das Problem der Abhängigkeit der Menschen von fossilen Energieträgern. Sie können lediglich helfen, den Prozess bis zum Ausgang der Ressourcen zu verzögern. Zusätzlich kann ein bewusstes Konsumverhalten der Menschen selbst Zeit gewinnen um neue Technologien entwickeln zu können.

1.4 Simulation

Ziel der Projektgruppe ist die hardwarebasierte Simulation energieautonomer Gebäude. Dazu sollen einerseits die Energieanlagen modelliert werden, andererseits eine Zentrale erschaffen werden, die eine intelligente Steuerung der Energieanlagen zur Strom- und Wärmeversorgung eines im Rahmen des Projekts definierten Gebäudes (ein öffentliches Gebäude, beispielsweise ein Hotel oder Krankenhaus) bereitstellt (vgl. Abbildung 1.4). Zu den Energieanlagen gehören Modelle von einer Photovoltaik-Anlage mit zusätzlichem Speicher, einem BHKW und unter Umständen eine Wärmepumpe. Die Zentrale beziehungsweise Steuerungsebene kommuniziert mit den Energieanlagen, erhält Zustandsinformationen und sendet Steuersignale. Die Steuersignale sind abhängig von äußeren Einflüssen, wie zum Beispiel Wetterdaten. Für die Generierung der Modelle der Energieanlagen werden im Folgenden die Betriebs-, Steuerungs- und Zustandsparamater von BHKW und Wärmepumpe definiert.

Der erste entscheidende Punkt ist die Dimensionierung der Energieanlagen. Die ist abhängig von den Gebäudedaten, in erster Linie der Strom- und Wärmebedarf und den dazugehörigen Lastkurven, die je nach Gebäudeart und unter Berücksichtigung der klimatischen Bedingungen und der Wetterdaten schwanken können. Mit Hilfe dieser Daten soll die Steuerung zur Maximierung der Effizienz optimiert werden.

1.4.1 Modellierung Mini-BHKW

Zu den Betriebscharakteristika eines BHKW gehört die Art des verwendeten Brennstoffs, die maximal aufnehmbare Antriebsenergie und die bei Volllast erreichbaren Wirkungsgrade für elektrische und thermische Energie. Weiterhin wichtig ist die zulässige Art der Steuerung. Kann das BHKW stufenlos reguliert werden oder gibt

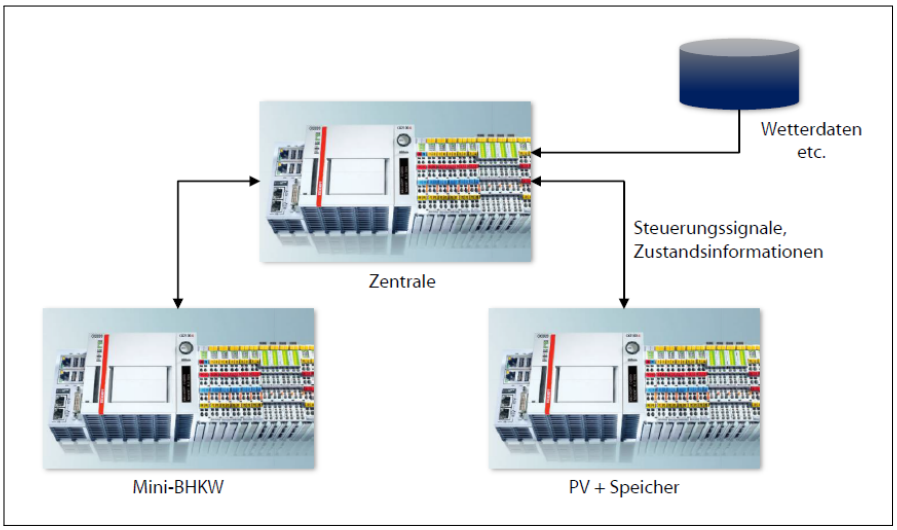


Abb. 1.4 Vereinfachter Systemaufbau

es vom Hersteller vorgegebene Steuerungsstufen? Wie beeinflusst der Betrieb in Teillast die elektrischen und thermischen Wirkungsgrade?

Die Betriebsparameter haben Einfluss auf das Steuerungsverhalten der Zentrale. Die Steuereinheit entscheidet erstmal über die Betriebsart des BHKW und sendet Sollwerte für die elektrische Leistung und die Wärmebereitstellung. Davon ist abhängig, mit welcher Leistungsaufnahme das BHKW zur jeweiligen Zeit läuft und wie sich die Zustandsparametern des BHKW verhalten.

Das Modell selbst muss Zustandsinformationen speichern und bereitstellen um die optimale Steuerung zu garantieren. Dazu zählen die aktuelle Leistungsbereitstellung, die Auslastung, die bisher erzeugte elektrische und thermische Leistung, die verursachte CO2-Emission, die vollen Betriebsstunden und für die Langzeiteffekte die Anzahl der Motorstarts und die zu berücksichtigen Wartungsintervalle des BHKW. Diese Parameter haben Einfluss auf das zukünftige Steuerungsverhalten und dienen der Analyse des Systems.

1.4.2 Modellierung Wärmepumpe

Zur Modellierung der Wärmepumpe sind folgende Betriebsparameter des Modells notwendig: Die Art der Wärmequelle, zum Beispiel Erdkollektoren oder Grundwasser, das verwendete Betriebsmittel, der Energieverbrauch und der Wirkungsgrad beziehungsweise Leistungszahl der Wärmepumpe.

Da die Wärmepumpe lediglich für die Bereitstellung von Wärmeenergie genutzt werden kann und nur geringe Flexibilität bezüglich der Leistungsaufnahme besteht, ist als einziger Steuerungsparameter der Sollwert für die Wärmebereitstellung innerhalb des Gebäudes notwendig. Danach wird entschieden, ob die Wärmepumpe läuft oder nicht. Eine Ausnahme bildet die Wärmepumpe mit 2-Verdichter-Betrieb, die hinsichtlich der Steuerung eine zusätzliche Modulationsstufe bietet.

Von der Steuerung abhängig sind die Zustandsinformationen des Modells. Sie geben Auskunft über die aktuell erzeugte und prognostizierte Wärmeenergie, sowie über die resultierte CO2-Emission. Des Weiteren sollen Statistiken über bisherige Laufzeiten zur Verfügung stehen um das System analysieren zu können.

1.4.3 Beispiel: BHKW im Krankenhaus

Krankenhäuser sind Gebäude, die über das ganze Jahr hinweg geheizt oder klimatisiert und mit Strom und Warmwasser versorgt werden müssen. Um die Versorgung der Bevölkerung zu gewährleisten, sind sie mit modernsten, aber auch energieintensiven Technologien ausgestattet. In der Realität wurden bisher nur wenige Blockheizkraftwerke in Krankenhäusern installiert, obwohl diese auf Grund ihrer spezifischen Verbrauchsstruktur, dem konstant hohen Grundlastbedarf an Elektrizität und Wärme, prädestiniert sind für die Versorgung dieser Art von Einrichtungen. Es gibt jedoch Projekte, in denen Krankenhäuser zur Senkung der Energiekosten durch BHKW versorgt werden sollen. Zum Ziel wurde sich gesetzt, die Gesamtkosten von Krankenhäusern zu senken mit dem positiven Nebeneffekt, die Umweltbelastung zu reduzieren.

Ein Beispiel ist das Projekt „Blockheizkraftwerke in Krankenhäusern - Kostensenkung durch effiziente Strom- und Wärmeerzeugung“ der Arbeitsgemeinschaft für sparsamen und umweltfreundlichen Energieverbrauch (kurz ASUE), welches den Weg zur Planung und Auslegung eine optimalen BHKW zur größtmöglichen Energieeinsparung und Emissionsverringerung beschriebt. Dieses Konzept kann für die Projektgruppe als Grundlage zur Dimensionierung des virtuellen BHKW dienen.

Das ASUE-Projekt unterteilt den Weg zum optimalen BHKW in vier Phasen von der Bestandsaufnahme bis zur Realisierung. In der Bestandsaufnahme werden die Daten des Krankenhauses erfasst und Wärme- und Strombedarf analysiert. Darauf folgt die Konzeption und Auslegung des BHKW mit Hilfe der Jahresdauerlinie des Wärmebedarfs, den Lastgang des Strombedarfs und der Wärme-, Strom- und Brennstoffbilanz. In der dritten Phase wird das BHKW hinsichtlich der Wirtschaftlichkeit untersucht und schlussendlich realisiert. Für die Simulation wichtig sind die ersten beiden Phasen, nach deren Durchlaufen das Modell erstellt werden kann.

Literaturverzeichnis

1. BGR: Energiestudie 2013. reserven, ressourcen und verfügbarkeit von energierohstoffen (2013)
2. EnergieAgentur.NRW: Arbeitsgemeinschaft für sparsamen und umweltfreundlichen energieverbrauch e.v. : Blockheizkraftwerke in krankenhäusern (2010). URL http://www.brennstoffzellen-heiztechnik.de/downloads/bhkwkrankenhaus-2010.pdf
3. Lord, N.W., Ouellette, R.P.: Heat pump technology. Ann Arbor Science (1980)
4. Thomas, B.: Mini-Blockheizkraftwerke: Grundlagen, Gerätetechnik, Betriebsdaten, 1 edn. Vogel Buchverlagl (2007)
5. Wosnitza Franz und Hilgers, H.G.: Energieeffzienz und Energiemanagement. Ein Überblick heutiger Möglichkeiten und Notwendigkeiten. Vieweg+Teubner Verlag (2012)

Kapitel 2
Gerätemodelle: Photovoltaik, elektrische Speicher

Marius Hacker

Zusammenfassung Diese Arbeit gibt einen Einblick in die Technik einer Photovoltaik-Anlage und einen Überblick über gängige elektrische Speicher. Sie stellt einen Bezug der Techniken zum Projekt „Hardwarebasierte Simulation energieautonomer Gebäude“ her und zeigt auf, welche Parameter für ein Modell zur Simulation von Photovoltaik-Analgen und elektrischen Speichern, wichtig sind.

2.1 Motivation

Steigende Strompreise und der Wunsch nach „sauberer“ Energie bewegen viele Menschen dazu über die Anschaffung einer Photovoltaik-Anlage nach zu denken. Oft wird der erzeugte Strom einer Photovoltaik-Anlage ins Stromnetz eingespeist und der benötigte Strom dann wieder aus diesem bezogen. Bei einem anderen Modell wird der erzeugte Strom zwischengespeichert und für den Eigenbedarf direkt genutzt. Die Projektgruppe, in deren Rahmen diese Arbeit geschrieben wurde, beschäftigt sich mit der Simulation eines solchen energieautonomen Gebäudes. Ein energieautonomes Gebäude kann z.B. durch eine Photovoltaik-Anlage mit Strom versorgt werden. Diese Seminararbeit befasst sich speziell mit Photovoltaik-Anlagen und elektrischen Speichern und stellt einen Bezug zum Projekt „Hardwarebasierte Simulation energieautonomer Gebäude“ her.

Im ersten Teil der Arbeit wird der Aufbau und die Funktionsweise einer Photovoltaikanlage genauer beschrieben, wichtige Kenngrößen eingeführt und anschließend auf die Anwendung im Projekt eingegangen.

Im zweiten Teil geht es um elektrische Speicher. Dort wird eine Übersicht über verschiedene Speicher, die Teilweise schon im Einsatz sind, gegeben und zu jedem Speichertyp die grobe Funktionsweise erläutert. Außerdem wird auch hier der Projektbezug hergestellt.

Carl von Oss ietzky Universität Oldenburg
E-mail: Marius.Hacker@uni-oldenburg.de

2.2 Photovoltaik

Eine Photovoltaikanlage wandelt die Strahlungsenergie der Sonne in Strom um. Dies geschieht mit Solarmodulen, deren typische blaue Oberfläche auf immer mehr Häusern zu sehen ist. Die Module einer Photothermieanlage, die die Sonnenenergie zum erhitzen von Wasser benutzt, werden in dieser Seminararbeit nicht behandelt.

2.2.1 Bestandteile einer Photovoltaikanlage

Eine Photovoltaikanlage besteht aus den Solarmodulen, einer Montagevorrichtung und einem Wechselrichter. Es werden meist mehrere Solarmodule zusammen geschaltet, um die gewünschte Fläche zu erhalten. Zur Befestigung der Solarmodule ist eine Montagevorrichtung nötig. Diese dient auch zur Ausrichtung der Module, sodass kleine Korrekturen möglich sind. Da die Solarmodule Gleichstrom liefern wird zur Nutzung im Haushalt und zur Einspeisung ins Stromnetzwerk ein Wechselrichter benötigt. Dieser wandelt den Gleichstrom in Wechselstrom um. [7] Viele Photovoltaik-Systeme haben zusätzlich noch einen Energiespeicher in Form eines Akkumulators angeschlossen. Dadurch kann die am Tag und überwiegend in den Mittagsstunden gewonnenen Energie gespeichert und z.B. in der Nacht benutzt werden. Der Akkumulator erfordert noch eine Ladeelektronik in Form eines Ladereglers. Dieser schützt den Akkumulator vor Überladung und Tiefentladung und verlängert so die Lebensdauer des Akkumulators. [7]

2.2.2 Geschichte/Entwicklung

Die Geschichte der Photovoltaik beginnt im Jahre 1860. Es sollte ein Transatlantikkabel verlegt werden. Um mögliche Fehler an der Leitung frühzeitig zu erkennen, wurde von dem damaligen leitenden Elektro-Ingenieur ein Prüfgerät entwickelt. Er verbaute in seinem Prüfgerät Stangen aus kristallinem Selen. Als das Prüfgerät nur nachts ohne Probleme funktionierte, erkannte der Elektro-Ingenieur Willoughby Smith, dass dies mit dem Sonnenlicht zusammenhing. 1876 wurde von William Grylls Adams und Richard Ebans Day festgestellt, dass in Selen ein Strom fließt, wenn Sonnenlicht darauf fällt. Einige Jahre später im Jahr 1885 kombinierte Charles Fritts mehrere Selen-Platten zu einem Solar-Panel zusammen. Dieses Solar-Panel war nur in der Forschung interessant, da es noch ein weiter Weg zu kommerziell nutzbaren Solarzelle war. 1905 versuchte Albert Einstein zu erklären, warum durch Einstrahlung von Sonnenlicht ein Strom in Selen fließt. Er erklärte diesen Effekt damit, dass Licht aus Energie-Teilchen besteht. Diese Energie-Teilchen sind heute als Photonen bekannt. 1953 wurde der Grundstein der heutigen größtenteils aus Silizium bestehenden Solarzellen gelegt, als Gerald Pearson, Darryl Chapin und Calvin Fuller erkannten, dass in Silizium ein viel stärkerer Stromfluss bei Lichteinfall ent-

steht als in Selen. Ein Jahr später präsentierten sie die erste anwendbare Solarzelle aus Silizium der Weltöffentlichkeit. [7]

2.2.3 Funktionsweise einer Solarzelle

Eine Solarzelle ist in mehreren Schichten aufgebaut. In Abbildung 2.1 sind diese in einem Querschnitt zu sehen. An der Vorderseite der Solarzelle befinden sich die Vorderseitenkontakte. Auf einer Solarzelle sind diese silbernen Metallstreifen leicht auf der blauen Oberfläche zu erkennen. Die markante blaue Oberflächenfärbung entsteht, da die Solarzellen mit einer Antireflexionsschicht überzogen werden. Durch diese Schicht geht weniger von der Sonnenenergie verloren. Unter dieser Schicht befindet sich zuerst die Negativ-Schicht und anschließend die Positiv-Schicht. In diesen beiden Schichten geschieht der eigentliche Effekt. Der Bereich zwischen den beiden Schichten wird Positiv-Negativ-Übergang oder kurz pn-Übergang genannt. An der Rückseite befinden sich noch die Rückseitenkontakte. [2]

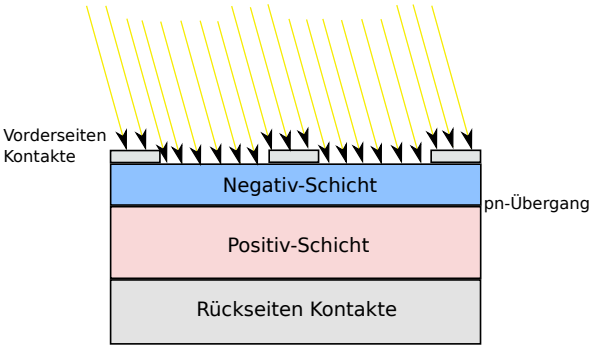


Abb. 2.1 Funktionsweise einer Solarzelle 1

Trifft Sonnenlicht auf die Positiv- und die Negativ-Schicht, werden in der Positiv-Schicht Elektronen frei gesetzt (In Abbildung 2.2 blau dargestellt) und hinterlassen Löcher (rot dargestellt). Kommt ein Elektron nah an den pn-Übergang wird dieses in die Negativ-Schicht bewegt. Die Löcher bleiben in der Positiv-Schicht. Dadurch entsteht in der Negativ-Schicht eine elektrische negative Ladung und in der Positiv-Schicht eine elektrische positive Ladung. Werden die Vorderseitenkontakte mit den Rückseitenkontakten verbunden und ein Verbraucher dazwischen geschaltet, so fließen die Elektronen über den Verbraucher zu den Löchern in der Positiv-Schicht und löschen sich gegenseitig aus. Es fließt ein Strom. [2]

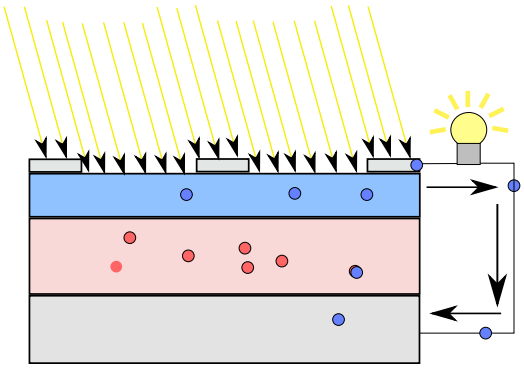


Abb. 2.2 Funktionsweise einer Solarzelle 3

2.2.4 Solarleistung Idealverlauf

Die erzeugte Energie einer Photovoltaik-Anlage hängt von der stärke der Sonneneinstrahlung ab. Am meisten Energie liefert eine Solarzelle, wenn das Sonnenlicht senkrecht auf sie fällt. Die Sonne liefert am meisten Energie in den Mittagsstunden, wenn sie senkrecht am Himmel steht. In Abbildung 2.3 wird ein Tagesverlauf einer Vorführanlage bei sonnigem Wetter gezeigt. Dieser ideale Verlauf kann nur bei wolkenlosem sonnigem Wetter entstehen. Besonders an wechselhaften Tagen entstehen deutliche Schwankungen bei der Energieerzeugung.

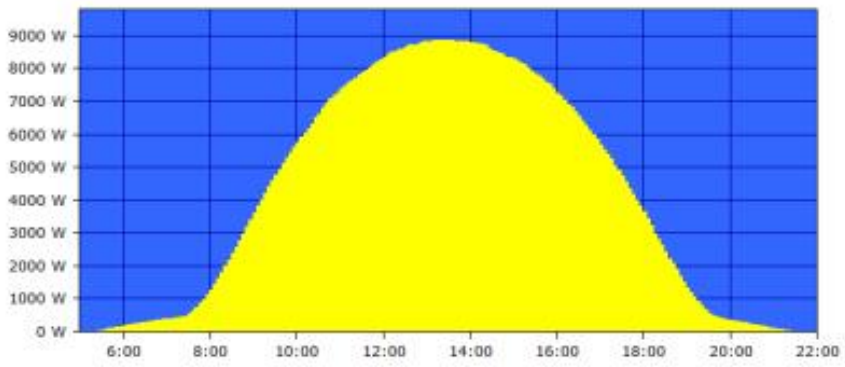


Abb. 2.3 Tagesverlauf bei Sonnigem Wetter[1]

2.2.5 Kenngrößen

Betrachten wir nun die wichtigen Kenngrößen einer Solarzelle. Ein wichtige Kenngröße zum Vergleich von Solarzellen ist der Wirkungsgrad. Er gibt an, wie viel der Leistung der Strahlungsenergie tatsächlich in elektrische Energie umgewandelt wird. Berechnet wird der Wirkungsgrad durch die Formel:

$$Wirkungsgrad\eta = \frac{P_{elektrisch}}{P_{licht}}$$

Wobei $P_{elektrisch}$ die elektrische Leistung und P_{licht} die Leistung der Strahlung ist. Polykristalline Silizium-Zellen erreichen einen Wirkungsgrad von ca. 14%. Bei Monokristallinen Silizium-Zellen sind es ca. 18% (max 24%). Die Leistung einer Solarzelle lässt sich mithilfe der Leistungsdichte der Sonne, der Solarfläche und dem Wirkungsgrad wie folgt berechnen:

P = Leistungsdichte der Sonnen · Solarfläche · Wirkungsgrad

[4]

Fallen die Sonnenstrahlen nicht senkrecht auf das Solarmodul, so muss noch die Neigung der Fläche mit eingerechnet werden:

P = Leistungsdichte der Sonnen · Solarfläche · Wirkungsgrad · Neigungsfaktor

Wie in [7] beschrieben, berechnet sich dieser Neigungsfaktor wie Folgt:

$$\text{Neigungsfaktor} = \cos \Theta$$

Wobei Θ der Winkel zwischen Normalenvektor der ebenen Fläche und dem Einfallswinkel der Sonnenstrahlung ist.

Eine oft angegebenen Kenngröße ist der Watt Peak. Er gibt die maximale Leistung eines Solarmoduls unter Laborbedingungen an.

2.2.6 Berechnung der Leistung einer Solarzelle

Wie im vorangegangenen Abschnitt beschrieben, wird die Leistung einer Solarzelle durch die Formel

P = Leistungsdichte der Sonnen · Solarfläche · Wirkungsgrad · Neigungsfaktor

berechnet. Betrachten wir nun die einzelnen Variablen. Für die Berechnung muss die Solarfläche bekannt sein. Sie wird üblicherweise in Quadratmetern angegeben und bei der Multiplikation mit der Leistungsdichte der Sonne, die üblicherweise in Watt pro Quadratmeter angegeben wird, ergibt sich die Sonnenleistung, die auf die gesammte Solarfläche trifft. [7]

Eine Solarzelle kann am meisten Strahlungsenergie umwandeln, wenn die Strahlung senkrecht auf sie trifft. Ist dies nicht der Fall, so müssen wir noch den Neigungsfaktor mit ein berechnen. Der Neigungsfaktor lässt sich wie folgt berechnen:

$$\text{Neigungsfaktor} = \cos \Theta$$

Θ ist der Winkel zwischen Normalenvektor der Solarfläche und dem Einfallswinkel der Strahlung. Ist der Einfallswinkel der Sonnenstrahlung nicht bekannt, kann dieser wie im Folgenden beschrieben berechnet werden.

Die Formel lautet wie folgt:

$$\begin{aligned}\cos\Theta = &\sin\delta \cdot \sin\varphi \cdot \cos s \\ &+ \sin\delta \cdot \cos\varphi \cdot \sin s \cdot \cos a \\ &+ \cos\delta \cdot \cos\varphi \cdot \cos s \cdot \cos\omega \\ &- \cos\delta \cdot \sin\varphi \cdot \sin s \cdot \cos a \cdot \cos\omega \\ &+ \cos\delta \cdot \sin s \cdot \sin a \cdot \sin\omega\end{aligned}$$

In die Formel müssen folgende Winkel eingesetzt werden:

φ	Geografischer Breite (Nord +, Süd -)
ω	Stundenwinkel (Vromittag +, Nachmittag -)
δ	Deklination (beschreibt jahreszeitliche Stellung der Erdachse)
s	Neigungswinkel zwischen Horizontale und Fläche (immer positiv)
a	Azimutwinkel der Fläche (Nord=0 Grad, Ost=90, Süd=180, West=270)

Die ersten drei Winkel sind nötig, um die Lage des Standortes der Solarzelle auf der Erde zur Sonnenstrahlung zu bestimmen. Die Winkel s und a beschreiben die Positionierung der Solarzelle. s gibt die Neigung der Solarzelle an. Wird die Solarzelle auf einem Dach befestigt, wäre die Neigung hier gleich der Neigung des Daches (z.B. 30 Grad). Der Azimutwinkel, hier mit a bezeichnet, gibt die Ausrichtung der Solarzelle an. Ist die Solarzelle Richtung Nord-Ost ausgerichtet beträgt der Winkel beispielsweise 45 Grad. Zum Schluss muss noch mit dem Wirkungsgrad der Solarzelle multipliziert werden und man erhält die tatsächliche Leistung der Solarzelle. Hierbei muss beachtet werden, dass bei der Berechnung davon ausgegangen wird, dass die Solarfläche gleichmäßig bestrahlt wird. [7]

2.2.7 Projektbezug

Bei der Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“ geht es um sich selbst mit Energie versorgende Gebäude. Eine Möglichkeit Energie zu erzeugen ist eine Photovoltaik-Anlage. Um den Bedarf an Energie zu decken, muss die Solarfläche groß genug ausgelegt sein. Zur Bestimmung der benötigten Solarfläche sind mehrere Parameter zu beachten. Die benötigte Solarfläche hängt von der Leistung der Sonne in der Region, dem Wirkungsgrad der Solarzelle, der benötigten Energie (hier vor allen Peak-Leistungen) und den eingesetzten Speichern ab. Die Leistungsstärke der Sonnen wird vom Deutschen Wetterdienst (DWD) [8] seit mehreren Jahren für Deutschland gemessen. Die Dimensionierung einer Photovoltaik-Anlage wird im Buch [2] von Andreas Wagner ausführlich beschrieben.

Da das Thema der Projektgruppe mit dem Simulieren von Photovoltaik-Anlagen zusammen hängt, betrachten wir nun die Parameter, die für ein Modell einer Photovoltaik-Anlage nötig sind. Um zu berechnen, welche Leistung eine Solarzelle liefert, wird der Wirkungsgrad, die Leistung der Sonnen und die Solarfläche benötigt. Das Modell wird in der Abbildung 2.4 dargestellt.

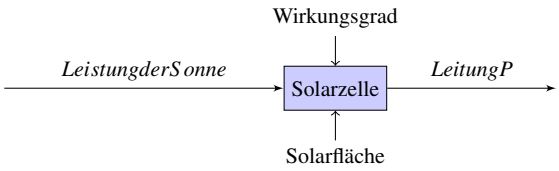


Abb. 2.4 Model Photovoltaik

2.3 Elektrische Speicher

Elektrische Speicher speichern elektrische Energie. Das speichern von elektrischer Energie ist immer dann nötig, wenn die Energie zu einem späteren Zeitpunkt benötigt wird. [2]

2.3.1 Kenngrößen

Um verschiedene Speicher vergleichen zu können, betrachten wir die wichtigen Kenngrößen von elektrischen Speichern. Der Wirkungsgrad ist eine dieser Kenngrößen. Er beschreibt das Verhältnis von aus dem Netz entnommener Energie zu ins Netz zurückgelieferter Energie:

$$Wirkungsgrad \eta = \frac{P_{ein}}{P_{aus}}$$

Bei einem Wirkungsgrad von 100% würde die gesamte elektrische Energie in eine andere Energieform umgewandelt werden. Ein weitere Kenngröße ist der Energieverlust. Der Energieverlust beschreibt den Energieverbrauch eines Speichers, wenn er weder entladen noch geladen wird. Mit dieser Kenngröße lässt sich leicht erkennen, ob der Speicher für die Speicherung von Energie über einen längeren Zeitraum geeignet ist, oder nur kurzzeitig effizient Energie speichern kann. Des weiteren kann ein Speicher eine gewisse Leistung liefern. [5]

2.3.2 Arten von elektrischen Speichern

Es gibt viele verschiedenen Arten von elektrischen Speichern. Elektrische Energie kann beispielsweise in Rotationsenergie, in einem magnetischen Feld oder auch als potentielle Energie gespeichert werden. Im Folgenden soll ein Überblick über die verschiedenen Techniken zur Speicherung elektrischer Energie gegeben werden.

2.3.2.1 Drehmassenspeicher

Der Drehmassenspeicher speichert Energie mithilfe eines Schwungrads. Das Schwungrad wird mithilfe eines Motors in Bewegung gesetzt, wodurch elektrische Energie in Rotationsenergie umgewandelt wird. Fügt man diesem System Energie hinzu, so erhöht sich die Drehzahl, entnimmt man Energie, so verringert sich die Drehzahl des Schwungrads. Beim Entladen wird der Motor in umgekehrter Richtung als Generator genutzt. [2]

Ein Schwungrad aus Metall kann mit einer Drehzahl von 5000 bis 10000 Umdrehungen pro Minuten betrieben werden. Bei Schwungrädern aus Verbundmaterialien sind Drehzahlen von bis zu 100000 Umdrehungen pro Minute möglich. [2]

Drehmassenspeicher haben einen Wirkungsgrad von 90% bis 95%. Da sie bis zu 20% der gespeicherten Energie in einer Stunde wieder verlieren, eignen sie sich nur zur Kurzzeitigen Speicherung von Energie. Ein moderner Drehmassenspeicher kann ungefähr eine Leistung von 3MW über einige Minuten liefern. Beim Einsatz im Stromnetz werden meist mehrere Drehmassenspeicher zusammengeschlossen. [2]

Die Investitionskosten für Drehmassenspeicher liegen mit 1000 bis 5000 Euro pro kWh recht hoch. [2]

2.3.2.2 Supraleitende magnetische Energiespeicher [2]

Beim Supraleitenden magnetischen Energiespeicher (kurz SMES) wird Energie in einem elektromagnetischen Feld gespeichert. Dazu wird eine Spule so stark abgekühlt, dass sie supraleitend ist. Anschließend wird solange Gleichstrom durch die Spule geleitet, bis sich ein stabiles Magnetfeld aufgebaut hat. Ist dies geschehen werden die Enden der Spule kurzgeschlossen und der Strom fließt nahezu verlustfrei in der Spule. Zum Entladen werden die beiden Enden wieder an einen Wechselrichter angeschlossen und der Strom kann wieder entnommen werden. Obwohl der Strom nahezu verlustfrei in den Energiespeicher eingespeist werden kann, haben SMES nur einen Wirkungsgrad von 90 bis 95%. Dieser Energieverlust entsteht, da die Spule die ganze Zeit über auf die Sprungtemperatur runter gekühlt werden muss, damit sie supraleitend bleibt. Die Kühlung sollte auch bei der Selbstentladung berücksichtigt werden. Es ergibt sich ein Verlust von 10 bis 12% pro Tag. Da größere Anlagen momentan nicht wirtschaftlich sind, werden zur Zeit nur kleine Anlagen im Bereich von bis zu 10 MW eingesetzt. Diese können Netzschwankungen im Sekundenbereich ausgleichen. [2]

2.3.2.3 Elektronische Doppelschichtkondensatoren

Ein Kondensator speichert Energie in einem elektrischen Feld. Dieses elektrische Feld entsteht, wenn durch anlegen einer Spannung positiv und negativ geladene Ionen aus dem Elektrolyt zu den jeweiligen Elektroden wandern. An der einen Elektrode entsteht eine positive und an der Anderen eine negative Ladung. Doppelschichtkondensatoren haben eine hohen Wirkungsgrad von über 98% und eine hohe Leistungsdichte, weswegen sie überwiegend als Kurzzeitspeicher eingesetzt werden. [2]

2.3.2.4 Elektrochemische Speicher

Akkumulatoren speichern elektrische Energie in chemischer Energie. Dabei besteht ein Akkumulator immer aus zwei Elektroden und einem Elektrolyt. Eine Elektrode wird durch eine chemische Reaktion positiv und die Andere Negativ geladen. Werden beide über einen Verbraucher verbunden, so fließt ein Strom. Diesen grundlegenden Aufbau findet sich in allen Akkumulatoren wieder. Es gibt viele verschiedene Akkumulatoren, die für verschiedene Einsatzgebiete sinnvoll sind. Im Folgenden werden die wichtigsten von ihnen vorgestellt. [2]

Blei-Säure-Akkumulatoren

Blei-Säure-Akkumulatoren werden, da sie mit 100 bis 300 €/ kWh nicht sehr teuer sind, in vielen Bereichen eingesetzt. Sie können kurzzeitig hohe Stromstärken abgeben und werden aus diesem Grund in Kraftfahrzeugen eingesetzt. Sie haben gegenüber anderer Akkumulatoren mit 30 Wh/kg eine geringe Energiedichte. [2]

Akkumulatoren auf Nickel-Basis

Es gibt drei wichtige auf Nickel-Basis basierende Akkumulatoren: Nickel-Cadmium (NiCd), Nickel-Metallhydrid (NiMH) und Nickel-Zink(NiZn). Da die NiCd Akkumulatoren das giftige Cadmium enthalten, werden überwiegend NiMH- und NiZn-Akkumulatoren verwendet. NiZn- haben gegenüber NiMH-Akkumulatoren eine 30 bis 50% höhere Energiedichte und können schneller geladen werden. Die Energiedichte von Akkumulatoren auf Nickel-Basis beträgt 40 bis 110 Wh/kg. Sie werden z.B. in tragbaren Geräten und Hybridfahrzeugen eingesetzt. [2]

Lithium-Ionen-Akkumulatoren

Lithium-Ionen-Akkumulatoren haben eine hohe Energiedichte und werden überwiegend in mobilen Geräten, wie z.B. Mobiltelefonen oder Digitalkameras, eingesetzt. Es werden Energiedichten von etwa 190 Wh/kg erreicht. [2]

Natrium-Nickelchlorid-Akkumulatoren

Natrium-Nickelchlorid-Akkumulatoren haben eine Energiedichte von 80 bis 90 Wh/kg. Die Betriebstemperatur einer Na-NiCl-Zelle liegt bei 300 ° C. [2]

Natrium-Schwefel-Akkumulatoren

Ein Natrium-Schwefel-Akkumulator erreicht eine Energiedichte von 130 Wh/kg. [2]

Redox-Flow Akkumulatoren

Ein großer Vorteil von Redox-Flow Akkumulatoren ist, dass sie praktisch keine Selbstentladung haben. Außerdem sind die gespeicherte Energie und die Leistung unabhängig von einander skalierbar. Redox-Flow Akkumulatoren weisen Energiedichten von 15 bis 25 Wh/kg auf und haben einen Wirkungsgrad von 70 bis 80 %. [2]

2.3.2.5 Pumpspeicherkraftwerke

Ein Pumpkraftwerk wandelt elektrische Energie in potentielle Energie. Beim Laden wird Wasser aus einem niedrig gelegenen Becken in ein höher gelegenes Becken gepumpt. Um die gespeicherte Energie wieder in elektrische Energie um zu wandeln, wird das Wasser von dem höher gelegenen Becken in das niedriger gelegene Becken abgelassen und treibt dabei einen Generator über eine Turbine an. Pumpspeicherkraftwerke können Energie über lange Zeiträume speichern, haben aber mit 75 bis 85% einen niedrigen Wirkungsgrad. [2]

2.3.2.6 Druckluftspeicherkraftwerke

Beim Druckluftspeicherkraftwerk wird elektrische Energie mithilfe von verdichteter Luft, die unter hohem Druck in einen Speicher gepresst wird gespeichert. Die Luft wird mithilfe eines Kompressors verdichtet. Zum Entladen, wird die Luft aus dem Speicher herausgelassen und treibt eine Turbine an, die über einen Generator Strom erzeugt. Bei der Verdichtung von Luft entsteht Wärme-Energie und wenn die

Luft wieder freigelassen wird kühlt sich die Luft ab. Durch diesen Wärmeverlust erreicht eine Druckluftspeicher nur einen Wirkungsgrad von etwas 50%. Wird diese Wärme zusätzlich mit einem Wärmespeicher gespeichert, dann kann ein Kraftwerk Wirkungsgrade von etwa 70% erreichen. [2]

2.3.3 Projektbezug

Bei der Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“ geht es um Gebäude, die sich selbst mit Strom versorgen. Bei der Nutzung erneuerbarer Energien treten zum einen Schwankungen in der Energieerzeugung auf, zum Anderen wird nicht den ganzen Tag über Strom erzeugt. Solarzellen liefern nur Tagsüber Energie und je nach Uhrzeit und Wetter nicht immer die volle Leistung.

Es wird nötig die Energie zu speichern. Es soll nun betrachtet werden, welche der vorgestellten Speichertechnologien sich für den Einsatz in Gebäuden eignen.

Drehmassenspeicher eignen sich zum Einsatz in Gebäuden eher wenig. Die Investitionskosten sind mit 30.000 bis 40.000 €pro kWh schon sehr hoch. Schwungräder liefern außerdem nur Energie für Sekunden bis Minuten und sind dadurch ungeeignet den Strombedarf über eine ganze Nacht zu decken. Supraleitende magnetische Energiespeicher haben zwar den Vorteil, dass sie elektrische Energie gut speichern können, aber sie sind wie auch Drehmassenspeicher aufgrund der Kosten und weil sie kurzzeitig Energie liefern ungeeignet. Beide Techniken wäre für den Einbau in ein Gebäude auch von der Größe her ungeeignet. Elektronische Kondensatoren können Energie nur Sekunden bis Minuten lang speichern. Sie eignen sich daher nicht als Langzeitspeicher, können aber zum abfangen von Spannungsschwankungen eingesetzt werden. Betrachten wir die Druckluft- und Pumpspeicher. Diese können Energie über einen längeren Zeitraum speichern. Pumpspeicher erfordern einen geeignete Umgebung und können nicht sinnvoll in Gebäuden genutzt werden. Genauso wie Druckluftspeicher, die eher ungeeignet zum Einbau in Gebäuden sind. Nur Akkumulatoren sind wirklich zum Einbau in Gebäuden geeignet. Sie haben eine hohe Energiedichte, benötigen somit nicht all zu viel Platz und können Energie für Stunden bis hin zu Tagen speichern. Ein weiterer wichtiger Aspekt sind die Investitionskosten die sich im Bereich von 200 bis 1.100 €pro kWh befinden.

In der Projektgruppe wird ein Modell eines elektrischen Speichers für die Simulation benötigt. Betrachten wir nun welche Parameter zur Modellierung eines elektrischen Speichers in Form eines Akkumulators nötig sind. Wie in Abbildung 2.5 dargestellt, werden drei Eingangsparameter benötigt: die Eingangsleistung, der Wirkungsgrad und der Energieverlust. Mit Eingangsleistung und Wirkungsgrad lässt sich bestimmen, wie der Speicher aufgeladen wird. Mit dem Energieverlust kann die Entladung des Speichers bei nicht Benutzung simuliert werden. Über diese Parameter kann die Ausgangsleistung bestimmt werden. Bei der Modellierung des elektrischen Speichers muss ebenfalls die maximale Kapazität des Akkumulators berücksichtigt werden.

In [3] wird für ein Energiespeicher folgende Gleichung aufgestellt:

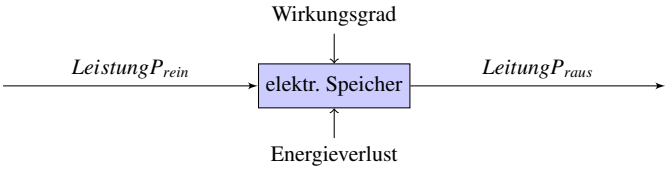


Abb. 2.5 Model elektrischer Speicher

$$Q'_\alpha = e_\alpha \cdot Q_\alpha$$

mit

$$e_\alpha = \begin{cases} e^+_\alpha & if Q_\alpha \geq 0 \\ 1/e^-_\alpha & else \end{cases}$$

Q_α ist die Energie, die dem System zugeführt wird und Q'_α ist die Energie, die das System tatsächlich aufnimmt. e_α ist der Wirkungsgrad beim Laden beziehungsweise Entladen. Dabei ist e^+_α der Wirkungsgrad beim Laden und $1/e^-_\alpha$ der beim Entladen.

Um die gespeicherte Energie zu einem bestimmten Zeitpunkt T zu ermitteln, muss der initiale Ladezustand des Speichers mit dem Integral über die Energie addiert werden. Wie in [3] beschrieben, sieht die Gleichung dann wie folgt aus:

$$E_\alpha(T) = E_\alpha(0) + \int_0^T Q'_\alpha(t)\,\mathrm{d}x$$

Mit diesem Modell kann ein einfacher elektrischer Speicher abgebildet werden. Bei diesem Ansatz werden Effekte die beim Entladen einer Batterie entstehen vernachlässigt. In [6] werden verschieden andere Modelle verglichen. Eine höhere Realitätsnähe liefert ein stochastisches Modell, hier angegeben mit 85%. Ein elektrochemisches Modell kann einen Akkumulator zu 99,9% abbilden, hat aber den Nachteil, dass es einen sehr hohen Rechenaufwand hat und für Echtzeit-Simulation nicht geeignet ist.

2.4 Fazit

Im ersten Teil dieser Ausarbeitung wurde die Photovoltaik-Anlage vorgestellt. Wird diese mit einem elektrischen Speicher, die im zweiten Teil der Arbeit vorgestellt wurden, kombiniert, so kann damit ein Gebäude über den ganzen Tag mit Strom versorgt werden. Da die Leistung der Sonnen nicht zu jedem Zeitpunkt konstant zur Verfügung steht und Solarzellen z.B. im Winter deutlich weniger Energie liefern, ist es sicherlich sinnvoll eine Photovoltaik-Anlage zusammen mit anderen Techniken zur Stromgewinnung zu betreiben.

Literaturverzeichnis

1. und Burs GbR, B.: Musteranlagen (2014). URL http://www.bredewald-elektro-heizung-sanitaer.de
2. Diekmann, B., Heinloth, K.: Energie. Springer (1997)
3. Geidl, M.: Integrated modeling and optimization of multi-carrier energy systems, vol. 68 (2007)
4. Löffler-Mang, M.: Optische Sensorik. Springer Science & Business (2011)
5. Prof. Dr.-Ing. W.-R. Canders, c.I.o.J.P.: Charakterisierung von Energiespeichern. Forschungsverbund Energie Niedersachsen Dezentrale Energiesysteme (2007)
6. Störzbach, A.: Emulation mobiler geräte: Integration eines batteriemodells. Studienarbeit: Universität Stuttgart, Institut für Parallele und Verteilte Systeme, Verteilte Systeme (2004)
7. Wagner, A.: Photovoltaik Engineering, vol. 3. Springer (2006)
8. Wetterdienst, D.D.: Monatliche strahlungskarten für deutschland (2014). URL http://www.dwd.de

Kapitel 3
Verfahren zur Eigenverbrauchsoptimierung

Michael Cordes

Zusammenfassung In der Vergangenheit war es für Eigentümer von dezentralen Stromerzeugern wie etwa Photovoltaikanlagen am wirtschaftlichsten, gewonnenen Strom möglichst vollständig in das Stromnetz einzuspeisen. Durch rechtliche Rahmenbedingungen wurde es in der Vergangenheit zunehmend attraktiver, die gewonnene Energie für den Eigenverbrauch zu nutzen. In dieser Arbeit werden unterschiedliche Verfahren zur Optimierung des Eigenverbrauches etwa durch entsprechende Stromspeicher oder Nachfragemanagement aufgezeigt sowie deren Nutzen analysiert. Für eine Maximierung des Eigenverbrauches ist etwa ein Heizstab sinnvoll, der überschüssigen Strom nutzt, um Warmwasser zu erzeugen. Gleichzeitig können Stromspeicher verwendet werden, die im Idealfall zusätzlich zu einer Entlastung des Stromnetzes durch Reduktion von Lastspitzen führen. Wesentlich bei einer optimalen Eigenstromnutzung ist die Integration unterschiedlicher Erzeuger und Speichertechniken auf Basis von Last- und Leistungsprofilen bzw. exakten Prognosen. Zu dieser Abstimmung können mathematische Optimierungsverfahren verwendet werden.

3.1 Einleitung

Aufgrund sich ändernder Rahmenbedingungen wird es zunehmend wirtschaftlicher, selbst erzeugten Strom auch selbst zu verbrauchen. Zudem können dadurch Investitionen in Stromnetze verringert werden. Daher werden in der folgenden Ausarbeitung verschiedene Strategien zur Optimierung des Eigenverbrauchs erläutert. Der Schwerpunkt liegt dabei auf Photovoltaikanlagen (im Folgenden kurz: *PV-Anlagen*) in Deutschland, dennoch lassen sich viele Konzepte auch auf andere Anlagentypen, z.B. Blockheizkraftwerke und andere Länder ausweiten. Zu beachten sind aber andere Fördervoraussetzungen. In der Literatur wird häufig eine Abgrenzung zwischen

Carl von Ossietzky Universität Oldenburg
E-mail: michael.cordes@uni-oldenburg.de

dem *Autarkiegrad* und dem *Eigenverbrauchsgrad* bzw. der *Eigenverbrauchsquote* vorgenommen [16] [27]. Der *Autarkiegrad* gibt die aus der Eigenerzeugung selbst verbrauchte Energie in Bezug auf den Haushaltsverbrauch an. Selbst verbrauchte Energie bezeichnet dabei die direkt genutzte oder über eine Batterie zwischengespeicherte Energie. Mit dem Eigenverbrauchsgrad ist die selbst verbrauchte Energie in Bezug auf die selbst erzeugte Energie gemeint:

$$\text{Eigenverbrauch} = \frac{\text{selbstverbrauchter PV-Strom}}{\text{erzeugter PV-Strom.}}$$

Im Folgenden wird die Definition des *Eigenverbrauches* verwendet, da diese auch im Erneuerbare-Energien-Gesetz Anwendung findet und daher bei der Betrachtung der Wirtschaftlichkeit eine wesentliche Rolle spielt [4, § 33 Abs. 1 Satz 1].

Die Ausarbeitung gliedert sich wie folgt: Zunächst wird in Kapitel 3.2 auf die Rahmenbedingungen in Deutschland eingegangen und z.B. die staatliche Förderung betrachtet. Anschließend werden in Kapitel 3.3 die unterschiedlichen Verfahren, mit denen der Eigenverbrauch konkret optimiert werden kann, dargestellt. Dabei wird auf Voraussetzungen der Eigenverbrauchsoptimierung eingegangen und das Potential durch Speichertechnik erläutert. Ferner wird kurz der Nutzen des Nachfragemanagements erörtert und beispielhaft ein möglicher, mathematischer Optimierungsansatz dargestellt. Zum Schluss werden in Kapitel 3.4 Auswirkungen der Eigenverbrauchsoptimierung aufgezeigt und deren Wirtschaftlichkeit beurteilt. Die Ausarbeitung endet in Kapitel 3.5 mit einem kurzen Fazit.

3.2 Rahmenbedingungen und rechtliche Grundlagen

Für eine Optimierung des Eigenverbrauches sprechen einige Rahmenbedingungen und gesetzliche Regelungen, die im Folgenden kurz dargestellt werden.

3.2.1 Auslastung der Netze

Der zunehmende Ausbau erneuerbarer Energien birgt zunehmend Probleme bei deren Integration in bestehende Stromnetze. Aktuell wird an manchen Wochenenden mit geringem Stromverbrauch und hoher PV-Erzeugung zwischenzeitlich die Hälfte der Last in Deutschland durch Solarstrom gedeckt [29]. Im Jahr 2013 wurden 23,4 % des Stromverbrauchs durch erneuerbare Energien gedeckt. Das Energiekonzept der Bundesregierung aus dem Jahr 2010 sieht vor, diesen Anteil bis 2030 auf 50 % zu steigern [2]. In Zukunft kann es daher z.B. an windigen und sonnigen Tagen häufiger zu Überschusssituationen kommen, in denen deutlich mehr Energie produziert als verbraucht wird. Diese überschüssige Energie kann teilweise an Orte transportiert werden, an denen noch Strombedarf herrscht oder andernfalls exportiert werden.

Dadurch, dass die Einspeisezeiten von PV-Anlagen einer starken Korrelation unterworfen sind, ist ein Abtransport häufig nur in höhere Spannungsebenen möglich [27]. Insgesamt sind somit hohe Investitionen in neue Stromtrassen erforderlich [5].

Um diese Investitionen möglichst gering zu halten bietet es sich an, die Stromerzeuger dazu zu animieren, den produzierten Strom selbst zu verbrauchen, da die maximale Belastung des Stromnetzes nicht der durch den Verbrauch, sondern durch die eingespeiste Leistung gegeben ist [30]. Dies sollte jedoch aus Nachhaltigkeitsaspekten nicht zu unnötigem Mehrverbrauch führen. Daher ist es sinnvoll, bestehende Lasten wo möglich in Zeiten hoher Erzeugung (z.B. mittags) zu verschieben und eventuelle überschüssige Energie für eine spätere Nutzung zwischenzuspeichern.

Im folgenden Abschnitt soll der im Zusammenhang mit PV-Anlagen wichtige Begriff der *Netzparität* erläutert werden.

3.2.2 Netzparität

Unter dem Begriff *Netzparität* versteht man den Zeitpunkt, zu dem die Stromgestehungskosten (hier: einer PV-Anlage) dem Strombezugspreis entsprechen. Jedoch lassen sich sowohl die Stromgestehungskosten als auch der Strombezugspreis unterschiedlich definieren. An dieser Stelle soll die am häufigsten verwendete Definition der Netzparität erläutert werden, bei der der Privathaushalt im Fokus steht. Dazu werden als Strombezugspreise die Haushaltsstrompreise für Privathaushalte betrachtet, die den Stromgestehungskosten (d.h. sämtlichen Kosten, die durch die Stromerzeugung entstehen) von PV-Aufdachanlagen gegenüberstellt werden. Dabei werden eine Rendite für den Betreiber sowie mögliche Finanzierungskosten nicht eingerechnet. [13]

Laut [32] liegen die Systempreise für Anlagen bis zu einer Größe von 10 kWp aktuell bei ca. 1800 EUR pro kWp (netto). Dadurch erzielen PV-Anlagen je nach Anlagentyp (Freifläche oder kleine Dachanlage) und Einstrahlung (zwischen 1000 und 1200 kWh/m^{2}a GHI in Deutschland) Stromgestehungskosten zwischen 7,8 und 14,2 Cent/kWh im dritten Quartal 2013 [20]. Diese liegen somit deutlich niedriger als der durchschnittliche Endkundenstrompreis von 29,38 Cent/kWh [8]. Die Einspeisevergütung betrug für PV-Anlagen, die im April 2014 in Betrieb gingen, je nach Anlagengröße zwischen 9,19 und 13,28 Cent/kWh für die kommenden 20 Jahre [32] (vgl. Kapitel 3.2.3.1). Dementsprechend ist für Neuanlagen dieser Größe die Netzparität erreicht: Der individuelle Vorteil ergibt sich aus der Differenz des Endkundenstrompreises und der Einspeisevergütung (letztere als Approximation der Stromgestehungskosten). Bei Altanlagen werden jedoch teilweise deutlich höhere Einspeisevergütungen gezahlt, sodass hier eine individuelle Betrachtung wesentlich ist.

3.2.3 Staatliche Förderung

Um das Potential der Eigenverbrauchsoptimierung zur Senkung der Investitionen in neue Stromtrassen nutzen zu können, gibt es verschiedene Fördermöglichkeiten, die im Folgenden kurz erläutert werden.

3.2.3.1 Förderung nach dem Erneuerbare-Energien-Gesetz (EEG)

Die Förderung der Einspeisung von Strom aus erneuerbaren Energien ist innerhalb des Erneuerbare-Energien-Gesetzes (im Folgenden kurz: *EEG*) durch Zahlungen von Einspeisevergütungen geregelt [4]. Dabei ist eine monatliche Degression der Einspeisevergütungen für neu angeschlossene Anlagen vorgesehen, um schrittweise die zunehmende Marktfähigkeit der Anlagen abzufedern. Diese Degression variiert je nach Zubau neuer Anlagen, bewegt sich aber um einen Wert von 1 % [4, § 20b]. Die Einspeisevergütungen werden den Anlagenbetreibern für eine Dauer von 20 Jahren garantiert [4, § 21 Abs. 2]. Für PV-Anlagen, die im April 2014 in Betrieb gingen, betrug die garantierte Einspeisevergütung je nach Anlagengröße zwischen 9,19 und 13,28 Cent/kWh [32]. Aufgrund der in Kapitel 3.2.2 dargestellten Situation, dass für viele Anlagenbetreiber mittlerweile der Zeitpunkt der *Netzparität* erreicht ist, spielt somit der Eigenverbrauch eine zunehmende Rolle.

PV-Anlagen, die zwischen dem Jahr 2009 und dem 31. März 2012 in Betrieb genommen wurden, können statt der Einspeisevergütung bei Eigenverbrauch eine Eigenverbrauchsvergütung erhalten. Ziel war, aufgrund des raschen Zubaus an PV-Anlagen einer Entlastung des Niederspannungsnetzes zu erreichen und dadurch Transportverluste und Kosten einzusparen [18]. Ab dem Jahr 2010 wurde die Nutzung der PV-Anlage zum Eigenverbrauch bei Neuanlagen auch wirtschaftlich attraktiv [10]. Ab Juli 2010 wurden zusätzlich Eigenverbrauchsquoten über 30 % höher vergütet [18].

Zum 01.04.2012 wurde aufgrund der Netzparität die Eigenverbrauchsförderung im EEG gestrichen [21]. Ferner gilt für ab dem 01.04.2012 ans Netz gegangene PV-Anlagen ab einer Leistung von 10 kWp (bis zu einer Leistung von 1 MWp), dass nur noch 90 % der erzeugten Leistung nach dem EEG vergütet werden. Die restliche Leistung muss soll entweder direkt vermarktet oder eigenverbraucht werden [4, § 33]. Zudem gilt seit dem 01.01.2014 (auch rückwirkend für Anlagen mit Installationsdatum ab 01.01.2012), dass die maximale Einspeisung am Verknüpfungspunkt mit dem Stromnetz auf 70 % der installierten Leistung begrenzt werden muss [4, § 6 Abs. 2 Satz 2b]. Demenstprechend darf jedoch der restliche Anteil der Generatorleistung eigenverbraucht werden. Eine Alternative stellt die Ausstattung der Anlage mit einer technischen Einrichtung dar, die es dem Versorger erlaubt, die Leistung abzuregeln [4, § 6 Abs. 2 Satz 2a i. V. mit § 6 Abs. 1]. Dies ist aufgrund der Installationskosten für kleinere Anlagen jedoch i.d.R. nicht wirtschaftlich. Zudem wird die Fernregulierung häufig über unidirektionale Rundsteueranlagen vorgenommen, die neben Sicherheitsrisiken zudem den Nachteil haben, dass ein vom Stromversorger gesendetes Abregelungssignal, welches als Broadcast an alle Anlagen gesendet

wird, nicht quittiert werden kann [22]. Dementsprechend ist der praktische Nutzen fragwürdig.

Am 8. April 2014 wurde der Gesetzesentwurf eines novellierten EEG vom Kabinett verabschiedet, ist damit jedoch aktuell noch nicht rechtskräftig. Das reformierte EEG soll zum 01. August 2014 in Kraft treten. Ziel der Reform ist es, den Kostenanstieg bei den erneuerbaren Energien spürbar zu senken. Dies soll durch eine für Neuanlagen um durchschnittlich 5 Cent/kWh niedrigere Einspeisevergütung über alle Technologien hinweg erreicht werden. Außerdem soll die Eigenstromerzeugung neu installierter Anlagen ab Überschreitung einer Bagatellgrenze an der EEG-Umlage (d.h. an der Finanzierung der Einspeisevergütungen) beteiligt werden. Die Bagatellgrenze ist im Entwurf auf 10 kWp installierte Leistung für höchstens 10 Megawattstunden selbst verbrauchten Stromes im Jahr festgesetzt. Die geplante Beteiligung für EEG- und KWK-Anlagenbesitzer beträgt 50 % der EEG-Umlage und somit aktuell 3,12 Cent/kWh. Für bestimmte Zielgruppen gelten Ausnahmeregelungen, die nachfolgend nicht weiter betrachtet werden. Die dargestellte, monatliche Degression der Einspeisevergütungen soll künftig geringer ausfallen. [3, § 58] [1] [14]

3.2.3.2 Förderrichtlinie für Batteriespeichersysteme

Um die technologische Entwicklung von Speichertechniken in Verbindung mit PV-Anlagen zu fördern, gilt seit dem 01. Mai 2013 die Förderrichtlinie für stationäre und dezentrale Batteriespeichersysteme in Kombination mit ab dem 01. Januar 2013 im Sinne des EEG installierten PV-Anlagen. Dadurch soll der Absatz dieser Speichersysteme erhöht und damit eine Kostensenkung bis hin zur Wettbewerbsfähigkeit errreicht werden. Die Förderung wird in Form von Tilgungszuschüssen zu Krediten der Kreditanstalt für Wiederaufbau (KfW) gewährt. Innerhalb des Programms werden Anlagen bis zu einer Leistung von 30 kWp gefördert. [7]

Ferner gelten u.A. die folgenden Voraussetzungen [7, Art. 5] [27, S. 16f.]:

- Am Netzanschlusspunkt darf die maximale Leistungsabgabe 60 % der installierten Leistung der PV-Anlage betragen. Diese Begrenzung gilt für die gesamte Lebensdauer der Anlage, mindestens aber 20 Jahre und ist nicht an den Speicherbetrieb gekoppelt.
- Der Wechselrichter muss über eine geeignete elektronische und offengelegte Schnittstelle zur Fernparametrierung bzw. Fernsteuerung verfügen. Ein Eingriff erfordert jedoch die Zustimmung des Anlagenbesitzers.

Die geförderten Kosten ergeben sich im Wesentlichen aus den Investitionskosten des Batteriespeichers inkl. der Installationskosten. Die spezifischen förderfähigen Kosten sind bei Neuinstallation einer Kombination aus Speicher und PV-Anlage auf 2000 EUR/kWp bzw. bei Nachrüstung eines Speichers auf 2200 EUR/kWp gedeckelt. Bei Nachrüstung eines Speichers gelten jedoch weitere Fördervoraussetzungen, auf die an dieser Stelle nicht weiter eingegangen wird. Die Förderhöhe ist auf 30 % der förderfähigen Kosten begrenzt, sodass eine maximale Förderung von 600 EUR/kWp

für ein kombiniertes System bzw. 660 EUR/kWp im Nachrüstungsfall möglich ist. [7, Art. 6] [27]

3.2.3.3 Fazit zur staatlichen Förderung

Zusammengefasst gelten für neu installierte PV-Anlagen, die nach dem EEG gefördert werden, strenge Auflagen an die maximale Einspeisung am Verknüpfungspunkt mit dem Stromnetz: Diese muss auf 70 % der installierten Leistung begrenzt werden. Daraus resultiert eine indirekte Verpflichtung zum partiellen Eigenverbrauch. Soll zusätzlich ein Batteriespeicher installiert und gefördert werden, muss die Eigenverbrauchsquote weiter erhöht werden, um eine Begrenzung auf 60 % der installierten Leistung der PV-Anlage zu erreichen. Gleichzeitig besteht aufgrund der Netzparität und monatlich weiter sinkender Einspeisevergütungen ein wirtschaftlicher Anreiz zur Erhöhung des Eigenverbrauchs. Dieser verliert jedoch etwas an Bedeutung, wenn das EEG wie geplant reformiert und die Eigenstromerzeugung an der EEG-Umlage beteiligt wird.

Im folgenden Kapitel werden unterschiedliche Verfahren dargestellt, mit denen sich eine Optimierung des Eigenverbrauches und damit auch etwa die Begrenzung der maximalen Einspeisung erreichen lässt.

3.3 Verfahren

Ohne den Einsatz von Speichersystemen oder Algorithmen zur Eigenverbrauchsoptimierung lassen sich bei PV-Anlagen je nach Anlagengröße Eigenverbrauchsquoten zwischen ca. 20 und 50 % erzielen [27]. Je größer die Anlagengröße, desto geringer der Eigenverbrauch, da bei konstantem Verbrauch kleine Anlagen einen größeren Anteil der benötigten Energie produzieren. Durchschnittlich werden Eigenverbrauchsquoten zwischen 20 und 34 % erreicht [13]. Dabei muss beachtet werden, dass die meisten Anlagen in der Vergangenheit nicht für hohe Eigenverbrauchsquoten optimiert wurden. Im folgenden Abschnitt wird zunächst auf Allgemeine Voraussetzungen eingegangen, die für die Optimierung des Eigenverbrauches wesentlich sind.

3.3.1 Allgemeine Voraussetzungen und Ziele der Eigenverbrauchsoptimierung

Die höchstmögliche Eigenverbrauchsquote wird erreicht, wenn der Stromverbrauch mit der Stromerzeugung zu jedem Zeitpunkt übereinstimmt. Um dies zu erreichen sind Last- und Leistungsprofile unabdingbar, die genauer in Kapitel 3.3.1.1 betrachtet

werden. Die Äquivalenz von Angebot und Nachfrage ist wesentlich, da Strom nicht ohne Verluste und hohe Kosten lagerbar ist [31].

Um die Stromnachfrage und das Stromangebot abzugleichen, gibt es im Wesentlichen 4 Möglichkeiten oder Kombinationen dieser Möglichkeiten:

- Die Unterstützung durch Speichertechnik (vgl. Kapitel 3.3.2), sodass in Zeiten hoher Erzeugung (z.B. in den Mittagsstunden) überschüssiger Strom gespeichert wird und zu einem späteren Zeitpunkt, etwa in den Abendstunden, zur Verfügung steht.
- Die Anpassung der Nachfrage an das Stromangebot (Nachfragemanagement, vgl. Kapitel 3.3.3.1).
- Mathematische Lösungsansätze, die eine möglichst optimale Lösung für den Abgleich von Angebot und Nachfrage errechnen und sich z.B. in Algorithmen des Nachfragemangements nutzen lassen. Diese Möglichkeiten werden kurz in Kapitel 3.3.3.2 aufgezeigt.
- Die Steuerung eines Blockheizkraftwerkes (kurz: *BHKW*). Der Autarkiegrad könnte erhöht werden, indem verschiedene, dezentrale Erzeugungsanlagen zu einem *virtuellen Kraftwerk* kombiniert werden. So könnte etwa eine PV-Anlage mit einem BHKW kombiniert werden: Während die PV-Anlage vor allem in den Sommermonaten tagsüber Strom liefert, könnte das BHKW im Winter genutzt werden. Das BHKW könnte zudem dann in Betrieb gehen, wenn unzureichende Sonneneinstrahlung herrscht, zudem verfügen BHKW in der Regel über Pufferspeicher für ihre erzeugte Energie und verhindern so Spitzenlasten [26]. Dennoch wäre in den Sommermonaten tagsüber ein Überangebot vorhanden, welches sich negativ auf den Eigenverbrauchsanteil auswirkt. Der Eigenverbrauchsanteil könnte jedoch u.U. dann erhöht werden, wenn auch Wärmeenergie einbezogen werden würde, die in BHKW genutzt werden kann. Dennoch wären für den überschüssigen Strom zusätzliche Speichertechniken dennoch sinnvoll. Die Steuerung eines BHKW wird in dieser Ausarbeitung nicht näher betrachtet.

Im folgenden Abschnitt wird der Nutzen von Last- und Leistungsprofilen näher dargestellt.

3.3.1.1 Last-/Leistungsprofile

Abbildung 3.1 zeigt eine vereinfachte, grob aufgelöste Lastkurve eines 4-Personen Haushaltes sowie die zugehörige Leistungkurve einer 5 kWp PV-Anlage. Die Abbildung verdeutlicht, wie sehr Stromverbrauch und Erzeugung voneinander abweichen: Während in den Mittagsstunden die PV-Anlage den höchsten Ertrag bringt, beträgt der Stromverbrauch zwischenzeitlich nur ein Viertel der Erzeugung. Zudem sind trotz der geringen Auflösung starke Schwankungen im Stromverbrauch erkennbar. In den Morgenstunden zwischen 7 und 8 Uhr sowie in den Abendstunden ab 16:30 Uhr ist der Verbrauch am höchsten. Der Ausgleich der bestehenden Abweichung von Erzeugung und Nachfrage resultiert in einem Optimierungsproblem, welches im Kapitel 3.3.3.2 näher dargestellt wird.

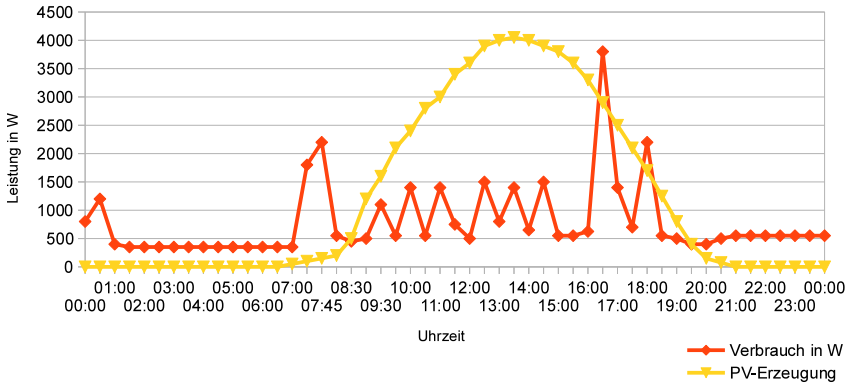


Abb. 3.1 Beispielhafter Verlauf von Erzeugung und Verbrauch an einem Sommertag am Beispiel eines größeren Haushalts (Stromverbrauch etwa 4900 kWh/a, 5 kWp PV-Anlage)
Quelle: Eigene Darstellung in Anlehnung an [10]

Um eine Planung im Voraus (z.B. für die jeweils nächste Viertelstunde) zu ermöglichen ist es notwendig, sowohl Stromverbrauch als auch -erzeugung möglichst genau zu prognostizieren. Diese Prognosen bilden auch die Grundlage für die optimale Nutzung eines Stromspeichers (vgl. Kapitel 3.3.2.3). Detaillierte Lastprofile, die auf Basis aggregierter Stromverbräuche von typischen Haushaltsgeräten den Verbrauch von Haushalten in zeitlich hoher Auflösung nachbilden, sind unter [13] zu finden. Diese leisten einen hohen Beitrag zur Eigenverbrauchsoptimierung: Je feiner Lastprofile aufgelöst sind und je besser sie den tatsächlichen Verbrauch wiederspiegeln, desto besser werden auch kurze Lastspitzen augezeigt. Um Lastkurven zu prognostizieren kann etwa die *Persistenz*-Strategie angewendet werden, die den Verbrauch des Wochentags der aktuellen Woche anhand des Verbrauchs des gleichen Wochentages der vorangegangenen Woche schätzt [27]. Sind zusätzlich möglichst genaue Prognosen über die Einspeisung verfügbar, wird ein Abgleich einfacher. Im Fall von Solaranlagen sind dafür genaue Daten über die aktuelle Wettersituation (z.B. Schneebedeckung im Winter) notwendig [9].

Aufgrund der in Kapitel 3.2.3 genannten Rahmenbedingungen wird es trotz Wirkungsgradverlusten zunehmend wirtschaftlicher, die gewonne Energie für den Eigenbedarf zu speichern.

3.3.2 Unterstützung durch Speichertechnik

Bereits die kleinsten PV-Anlagen erzeugen Überschussstrom, der nicht ohne weiteres nutzbar ist [17]. Daher sollen zunächst unterschiedliche Technologien vorgestellt und kurz evaluiert werden.

3.3.2.1 Thermische Speichersysteme

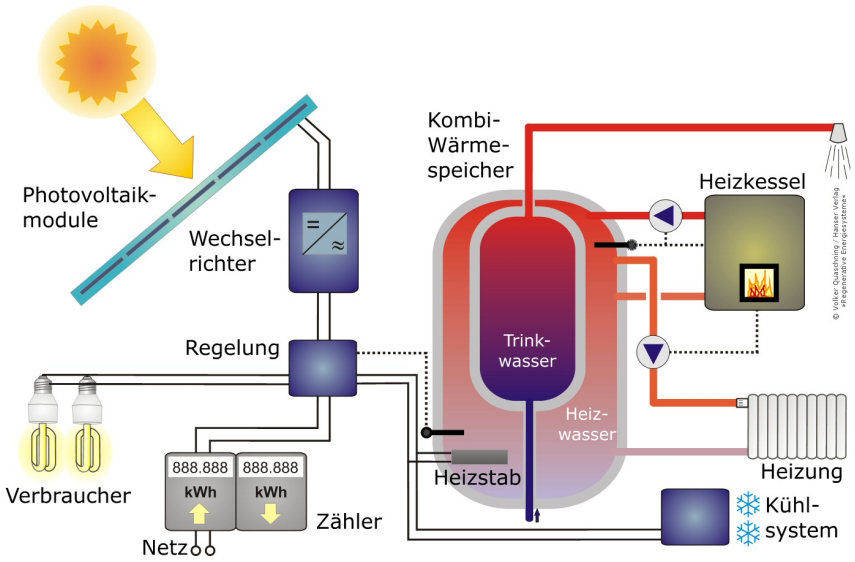


Abb. 3.2 Netzgekoppeltes PV-System mit Heizstab zur Erhöhung des Eigenverbrauchsanteils
Quelle: [23]

Der größte Teil des Energieverbrauches in Haushalten wird für den Bereich der Raumwärme (bei Bürogebäuden auch Kühlung) sowie für die Brauchwasserbereitstellung verwendet [28]. Kurzfristige Wasserspeicher sind in fast allen Haushalten vorhanden. Dementsprechend besteht eine Möglichkeit der Eigenverbrauchsoptimierung darin, überschüssigen Strom in Wärme zu wandeln und für die Erwärmung von Wasser oder Gebäuden zu verwenden. Dazu kann ein einfacher Heizstab verwendet werden, der überschüssige PV-Energie zur Erhitzung des bereits vorhandenen Wärmespeichers verwendet. Ein beispielhafter Aufbau eines Systems mit einem Heizstab ist in Abbildung 3.2 dargestellt. Diese Nutzung ist gekennzeichnet durch niedrige Investitionskosten sowie eine einfache Integrierbarkeit [17]. Ferner lassen sich damit sehr hohe Eigenverbrauchsquoten realisieren (vgl. Kapitel 3.3.2.2). Ein Nachteil

ist jedoch, dass sich im Jahresverlauf Angebot und Nachfrage gegenläufig entwickeln, da im Winter mehr Wärmeenergie nachgefragt wird, während im Sommer mehr gewonnen wird. Wird der Heizstab lediglich für die Warmwasseraufbereitung verwendet, spielen jahreszeitliche Unterschiede jedoch kaum eine Rolle [30]. Alternativ kann in eine Wärmepumpe investiert werden, welche elektrisch betrieben wird und in der Regel mit einem Pufferspeicher ausgerüstet ist [30].

Ebenfalls von Nutzen für den Ausgleich täglicher Schwankungen des Energiebedarfs und -angebots (Tag/Nacht-Zyklus) sind kurzfristige Stromspeichersysteme, die im folgenden Abschnitt vorgestellt werden.

3.3.2.2 Kurzfristige Stromspeichersysteme

Um den Nutzen kurzfristiger Stromspeichersysteme analysieren zu können, sind möglichst hoch aufgelöste Prognosen notwendig, um auch sehr kurzfristige Lastspitzen zu erfassen. Zum Ausgleich der täglichen Schwankungen und damit des Tag/Nacht-Zyklus ist laut [11] für eine 5 kWp PV-Anlage und einen Vier-Personen-Haushalt eine Speicherkapazität zwischen zwei und 34 kWh erforderlich.

Würde der Kurzzeitspeicher mit einem Langzeitspeicher kombiniert, könnte durch den Langzeitspeicher täglich eine konstante Offset-Leistung im Bereich von -700 W bis 1700 W abgedeckt werden. Darunter ist eine errechnete, planbare Grundleistung zu verstehen, die für den jeweiligen Tag konstant bereitgestellt wird, im Jahresverlauf aber im genannten Intervall schwankt. Im Sommer sollte diese Offset-Leistung überwiegend positiv sein (Aufladung), im Winter überwiegend negativ (Einspeisung). Aufgrund der begrenzten Kapazität sind jedoch auch im Winter positive und im Sommer negative Offset-Leistungen erforderlich. Diese Grundleistung kann die benötigte Kapazität des Kurzzeitspeichers auf einen Bereich von 0,4 kWh bis 17 kWh reduzieren [11]. Langfristige Speicher wären vor allem dann sinnvoll, wenn eine Maximierung des Eigenverbrauches auch bei mehreren oder größeren, volatilen Erzeugern mit saisonal schwankender Erzeugung gewünscht ist. Diese sind jedoch aufgrund der benötigten Kapazität mit hohen Investitionskosten sowie einem hohen Platzbedarf verbunden und werden daher in dieser Ausarbeitung nicht näher betrachtet [19]. Für gewerbliche Gebäude, z.B. Hotels oder Krankenhäuser, könnten diese langfristigen Speicher aber auch als unterbrechungsfreie Stromversorgung dienen und somit zusätzliche Sicherheit bei einem Stromausfall geben.

In [19] werden die Kosten verschiedener Speichertechnologien miteinander verglichen: Während bei einer geringen Auslastung (d.h. wenige Lade- und Entladezyklen) Bleibatterien am günstigsten sind, werden die geringeren Investitionskosten bei zunehmender Zyklenzahl durch höhere Lebensdauern anderer Systeme kompensiert. Den Lithium-Ionen-Batterien wird aufgrund hoher erwarteter Lerneffekte und einem daraus resultierenden hohen Kostensenkungspotential die größte Bedeutung zugeschrieben. Die am Markt zu findenden Lösungen bestätigen dies [12]. Dabei wurden technologiespezifische Eigenschaften der Speichersysteme (z.B. die erwartete Lebensdauer in Zyklen, Betriebs- und Wartungskosten, uvm.) in die Berechnung einbezogen, auf die im Folgenden nicht detaillierter eingegangen wird.

Andere Technologien sind hauptsächlich für langfristige Speichersysteme geeignet. Der Systempreis (d.h. schlüsselfertig inkl. Installation, Batteriemanagement und Zubehör, jedoch ohne Mehrwertsteuer) lag Ende Juni 2013 bei durchschnittlich rund 2000 EUR/kWh Nennkapazität für Li-Ionen-basierte Systeme verglichen mit 1500 EUR für Blei-basierte Systeme [12]. Gleichzeitig ist es für Kurzzeitspeicher wichtig, dass die Batterien über eine schnellstmögliche Lade- und Entladezeit verfügen, um kurzfristig Energie bereitstellen bzw. aufnehmen zu können. Bei Hochleistungs-Lithiumbatterien ist bereits eine Entladezeit von 2 bis 5 Minuten möglich - auch langsameres Entladen stellt jedoch kein Problem dar [28]. Aufgrund der erwarteten hohen Ausnutzung der Batterie durch die Eigenverbrauchsoptimierung wird im Weiteren die Lithium-Ionen-Technologie betrachtet.

Welche Speicherkapazität im Einzelnen optimal ist hängt davon ab, in wie weit die Batterie im Sommer entladen wird: Ein zu großer Speicher würde bei konventioneller Betriebsführung dazu führen, dass der Speicher im Tagesverlauf nicht bis zum Folgetag entladen werden kann [33]. Eine zu geringe Speicherkapazität resultiert bei gleicher durchgesetzter Energiemenge in einer höheren Zyklenzahl, welche die Lebensdauer reduziert [12]. Um einen hinreichend großen Nutzen zur Stabilisierung der Volatilität des Stromnetzes zu erzielen, sollte das Batteriesystem mit einer netzdienlichen Betriebsführung betrieben werden (vgl. Kapitel 3.3.2.3). Diese Netzdienlichkeit ist auch Voraussetzung für eine Subventionierung durch die Speicherförderrichtlinie der KfW, die eine Begrenzung der Anschlussleistung vorsieht (vgl. hierzu Kapitel 3.2.3.2).

Tabelle 3.1 Eigenverbrauchsanteile verschiedener Kombinationen aus PV, Batterie und Heizstab (Strategie: lineare Optimierung). Datenquelle: [17]

		Installierte PV-Leistung				
Eigenverbrauch (%)		**1 kW**	**2 kW**	**3 kW**	**5 kW**	**10 kW**
Batterie	**Ohne**	82	63	52	38	23
	1 kW	95	79	65	47	27
	3 kW	97	90	77	56	33
	5 kW	97	93	84	63	38
Heizstab	**5 kW**	100	100	98	83	60

In wie weit sich der Eigenverbrauch bereits durch kleine Speicher optimieren lässt, ist in Tabelle 3.1 dargestellt. Die Werte wurden dabei auf Basis realer Stromlastgänge aus Smart Metern in der Nähe von Nürnberg ermittelt und zehn zufällige Haushalte mit einem Jahresverbrauch von 5800 bis 6000 kWh ausgewählt [17]. Der Einsatz der PV-Anlage, der Speicher (Batteriespeicher mit Lithium-Ionen-Technologie), des Netzbezuges sowie des Heizstabes wurde mithilfe einer linearen Optimierung (mit dem Ziel der bestmöglichen Wirtschaftlichkeit) in GANS/CPLEX bestimmt. Die ermittelten Werte sind aufgrund der linearen Optimierung jedoch nicht direkt übertragbar. Zudem sind die Haushalte durch einen hohen Stromverbrauch (entspricht etwa dem Mittelwert von 5-Personen-Haushalten [15]) gekennzeichnet. Dennoch ist erkennbar, dass sich vor allem in Verbindung von kleinen Anlagen mit kleinen Batterien hohe Eigenverbrauchsanteile realisieren lassen. Bei einer PV-Anlage mit

einer Leistung von 10 kWp kann selbst bei einer Speichergröße von 5 kWh lediglich 38 % eigenverbraucht werden. Dies führt zu der Annahme, dass etwa die Inanspruchnahme der KfW-Speicherförderrichtlinie hauptsächlich bei Kleinanlagen möglich ist.

Interessant sind ferner die erzielbaren Eigenverbrauchsquoten bei Verwendung eines Heizstabes (5 kW): Bei kleinen Anlagen könnten im Idealfall bis zu 100 % Eigenverbrauch erzielt werden. Bei einer 10 kW PV-Anlage führt der Heizstab zu einer Erhöhung des Eigenverbrauchsanteils um 37 Prozentpunkte. Der finanzielle Nutzen ist dabei abhängig von den eingesparten Brennstoffkosten. Wesentlich ist zudem die Wahl einer korrekten Strategie zur Steuerung eines Akkumulators, auf die im nächsten Abschnitt eingegangen wird.

3.3.2.3 Betriebsführungsstrategien

Eine wichtige Rolle bei der Verwendung von Speichertechnologien spielt nicht nur dessen Auswahl, sondern die Integration in ein Anlagenkonzept [12]. Dazu existieren verschiedene Betriebsführungsstrategien mit jeweils unterschiedlichen Zielen. In der Literatur werden im Wesentlichen zwei Strategien unterschieden [34]:

- *Konventionelle Betriebsführung*: Sobald die Erzeugungsleistung höher als der Verbrauch ist, wird der überschüssige Strom möglichst vollständig zur Aufladung der Batterie genutzt. Dies kann dazu führen, dass diese bereits vor den erzeugungsintensiven Mittagsstunden vollständig geladen ist. In den Zeiträumen, wo der Verbrauch höher als die Einspeisung ist, wird vorrangig die gespeicherte Energie verwendet. Diese Strategie dient der Maximierung des Eigenverbrauchs, hat aber keine netzdienlichen Effekte, da Lastspitzen kaum minimiert werden.
- *Netzdienliche Betriebsführung*: Für eine bestmögliche, netzdienliche Betriebsweise muss die Annahme getroffen werden, dass die solare Einstrahlung und der Stromverbrauch perfekt prognostiziert werden können. Die Batterie wird dann im Zeitraum der höchsten solaren Einstrahlung geladen, um eine hohe Belastung des Stromnetzes durch die Einspeisung zu vermeiden. In den Zeiträumen, wo der Verbrauch die Einspeisung übersteigt wird ebenfalls vorrangig gespeicherte Energie verwendet. Mithilfe dieser Strategie kann die Einspeisespitze - bei perfekten Prognosen - um etwa 40 % reduziert werden. Dadurch könnten bei gleicher Netzbelastung 66 % mehr PV-Anlagen mit Batterieunterstützung installiert werden, wenn sich diese ebenfalls netzdienlich verhalten und ggfs. eine Fernregulierung zulassen. Ferner kann sichergestellt werden, dass die PV-Generatorleistung auf einen Wert zwischen 60 und 70 % reduziert werden kann. Die Residuallast, d.h. die Stromnachfrage, die von regelbaren Kraftwerken gedeckt werden muss, würde stabilisiert und somit besser planbar.

Die unterschiedlichen Strategien werden in Abbildung 3.3 grafisch gegenüberstellt. Die Betriebsführungskonzepte lassen sich insbesondere bei der Verwendung mehrerer Speicher weiter optimieren, vgl. z.B. [11].

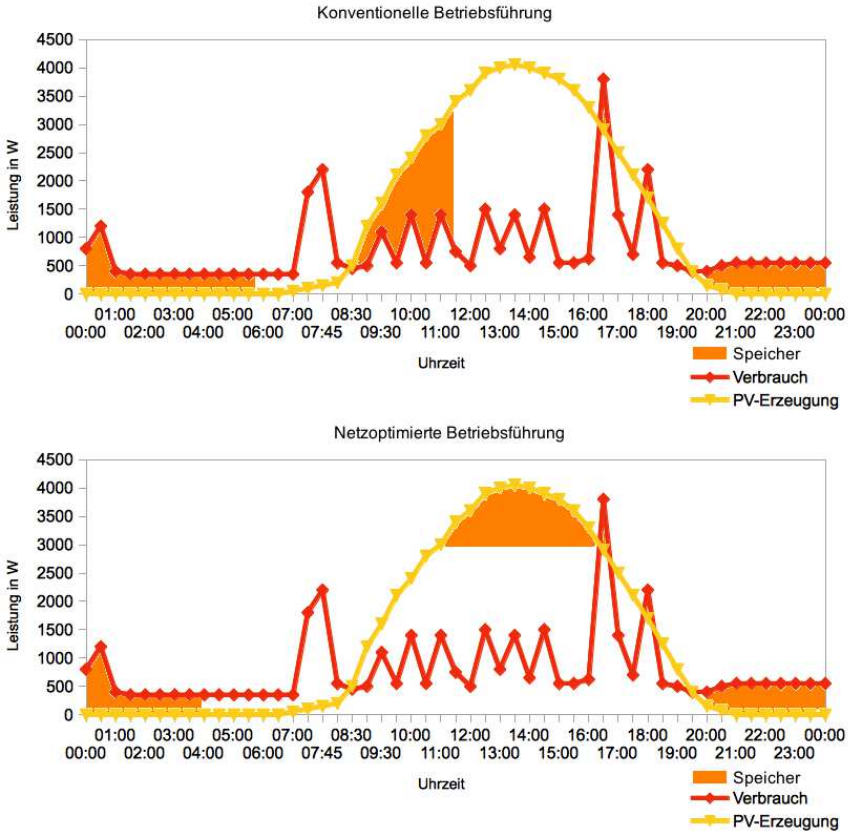


Abb. 3.3 Betriebsführungsstrategien im Vergleich
Quelle: Eigene Darstellung in Anlehnung an [34]

3.3.3 Weitere Optimierungspotentiale

3.3.3.1 Nachfragemanagement

Neben der Steuerung des Angebots besteht außerdem die Möglichkeit, die Nachfrage (in Grenzen) zu steuern. Unter den Begriffen *Lastmanagement* bzw. *Demand Side Management (DSM)* werden alle Maßnahmen zusammengefasst, die den Stromkonsum auf Seiten des Verbrauchers beeinflussen. Es besteht etwa die Möglichkeit, dem Verbraucher mithilfe von Preissignalen zu animieren, seine Stromnachfrage in Zeiten höheren Angebots zu verschieben. Die Reaktion der Verbraucher auf solche Anreize wird als *Demand Response (DR)* bezeichnet, häufig werden die Begriffe jedoch synonym verwendet. [25]

In Bezug auf PV-Anlagen besteht eine einfache und kostengünstige Variante von DSM darin, Lasten etwa per Zeitschaltuhr in den Zeitraum der höchsten Erzeugung zwischen 11 und 16 Uhr zu verschieben [13]. Dies stellt eine Variante des *Load shifting* dar [35] (vgl. Abbildung 3.4).

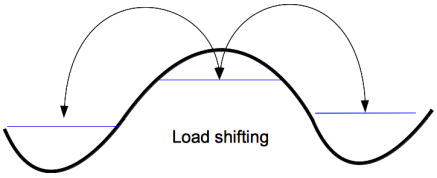


Abb. 3.4 DSM-Strategie Load Shifting
Quelle: Eigene Darstellung in Anlehnung an [35]

Bestehende, klassische Stromzähler sollten durch *Smart Meter* ersetzt werden [13]: Diese sollten in der Lage sein, sowohl die Erzeugungslage als auch den momentanten Verbrauch verständlich zu illustrieren. Damit wird dem Verbraucher die Möglichkeit gegeben, seinen Verbrauch manuell je nach Erzeugungslage anzupassen und somit den Eigenverbrauch besser abschätzen zu können. Dies kann durch Preisanreize ergänzt werden, indem z.B. die Einspeisevergütung in Zeiten höherer Erzeugung geringer ausfallen könnte. Dazu müsste jedoch das EEG entsprechend angepasst werden.

Alternativ kann eine *direkte Steuerung* einzelner Geräte vorgenommen werden. Dies kann z.B. einen Kühl- oder Gefrierschrank betreffen, die mit der Kühlung beginnen, sobald die Temperatur im Innenraum eine Obergrenze erreicht hat. In der Mittagszeit (bei hoher PV-Erzeugung) kann die Temperatur dann weiter als gewöhnlich heruntergekühlt werden, sodass in Zeiten geringerer Erzeugung eine Zeit lang auf Energiezufuhr verzichtet werden kann. Der Kühlraum dient somit als impliziter Speicher. [25]

Ebenfalls ließen sich z.B. Waschmaschinen so programmieren, dass ein spätester Endtermin für das Wäschewaschen eingegeben wird. Daraufhin könnte die Waschmaschine selbstständig (z.B. anhand prognostizierter Stromerzeugung) entscheiden, wann der Waschgang ausgeführt wird. Für die direkte Steuerung sind neben obligatorischen Prognosen auch Kommunikationseinrichtungen notwendig, die die Geräte über die aktuelle Erzeugungslage informieren. Diese Einrichtungen sollten bidirektional ausgelegt sein, um eine Rückmeldung der Geräte und somit genauere Verbrauchsprognosen zu ermöglichen. Eine Möglichkeit diese direkte Steuerung zu realisieren wären verbesserte *Smart Meter*, die Haushaltsgeräte mit entsprechenden Daten versorgen und aus den Rückmeldungen Prognosen errechnen können. Eine zusätzliche Regelungseinheit, die diese Aufgaben übernimmt, stellt eine weitere Möglichkeit dar. Ebenfalls kann eine *Spitzenlastbegrenzung* sinnvoll sein. Bei dieser DSM-Variante wird die aktuelle Spitzenlast überwacht. Ist diese höher als gewünscht, wird mithilfe eines Maximum-Wächters eine Leistungsbegrenzung vorgenommen und geeignete Lasten abgeschaltet [25]. Dies kann für die Eigenverbrauchsoptimierung von Nutzen sein, um etwa die Spitzenlast in Zeiten geringer bzw. keiner Erzeugung zu begrenzen oder um die Spitzenlast auf die maximal gewünschte Batterielast anzupassen, sofern Energie aus einer Batterie bezogen wird. Eine Anwendung erscheint jedoch vor allem im gewerblichen Bereich sinnvoll, da das Potential abschaltbarer Lasten in Privathaushalten vergleichsweise gering ist.

Die Potentiale, die sich durch Nachfragemanagement (auf Basis direkter Steuerung und Verschiebung energieintensiver Verbraucher um ein bis drei Tage) ergeben, liegen gegenüber dem Status Quo (ca. 20 bis 34 % Eigenverbrauchsanteil) bei einer Verbesserung um etwa 5 bis 11 Prozentpunkte. Jedoch konnte bereits durch eine manuelle Lastverschiebung energieintensiver Verbraucher auf den Zeitraum zwischen 11 und 16 Uhr eine Verbesserung um 3 bis 7 Prozentpunkte erreicht werden, sodass die zusätzlichen Kosten der Kommunikationseinrichtungen und der Komfortgewinn mit dem zusätzlich erzielbaren Eigenverbrauchsanteil genau abgewogen werden müssen. [13]

3.3.3.2 Mathematische Lösungsansätze

Aufgrund der vielfältigen Möglichkeiten, den Eigenverbrauch zu optimieren ergibt sich ein Optimierungsproblem, bei dem der Schwerpunkt darauf liegt, zu jedem Zeitpunkt die Erzeugung und die Nachfrage auf Haushaltsebene möglichst vollständig auszugleichen. Das Optimierungsproblem bei Verwendung eines Speichers und einer PV-Anlage lässt sich wie folgt beschreiben (auf Grundlage von [24]):

$$min(|P_{GRID}(t)|) \ \vee \ min(P^2_{GRID}(t)) \tag{3.1}$$

$$P_{GRID}(t) = P_{PV}(t) + P_{BAT}(t) + P_{LOADS}(t) \tag{3.2}$$

$$P_{GRID}(t) \leq P^{max}_{GRID}(t) \tag{3.3}$$

$$SOC^{min} \leq SOC(t) \leq SOC^{max} \tag{3.4}$$

$$P^{min}_{BAT} \leq P_{BAT}(t) \leq P^{max}_{BAT} \tag{3.5}$$

$$SOH(t) \geq SOH^{min} \tag{3.6}$$

Dabei steht P für die Leistung und SOC für den aktuellen Ladezustand. SOH steht für die verbleibende Kapazität des Akkus in Bezug zur ursprünglichen Kapazität und spiegelt somit den Zustand des Akkumulators wieder. P_{PV} gibt somit die Einspeisung der PV-Anlage an (immer negativ oder null) während P_{BAT} die Leistung der Batterie (negativ bei Einspeisung, positiv bei Aufladung) wiederspiegelt. P_{LOADS} steht für den aktuellen Stromverbrauch (immer positiv bzw. null). P_{GRID} steht dementsprechend für einen Strombezug (positiv) oder eine Einspeisung ins Stromnetz (negativ). Das Minimierungsproblem in Gleichung 3.1 ergibt sich somit daraus, dass im Optimalfall, wenn sich Angebot und Nachfrage vollständig ausgleichen, die Summe der Leistungen der PV-Anlage, der Batterie und des Verbrauchs Null ist (vgl. Gleichung 3.2). Desweiteren gelten einige Nebenbedingungen, etwa darf die Leistung am Netzanschlusspunkt eine definierte Maximalleistung nicht überschreiten (vgl. Gleichung 3.3). Der Ladezustand des Akkumulators sollte, um eine optimale Lebensdauer zu gewährleisten, in einem definierten Intervall liegen und etwa einen minimalen Ladezustand nicht unterschreiten (Gleichung 3.4). Gleiches gilt für die Lade- bzw. Entladeleistung des Akkus, die ebenfalls begrenzt werden sollte (Gleichung 3.5). Unterschreitet der Akku eine Minimalkapazität ist er defekt und muss ersetzt werden (Gleichung 3.6).

Das dargestellte Problem lässt sich auch für weitere Anlagen (z.B. BHKW) und Speicher erweitern. Da die Erzeugung und die Last schwankt, sind genaue Prognosen darüber unabdingbar. Gleichzeitig muss ein zeitlicher Abstand zwischen den Berechnungsschritten zur Ermittlung der Inputwerte gewählt werden, der möglichst gering ausfallen sollte. Zur Lösung eines solchen nichtlinearen Optimierungsproblems existieren verschiedene Verfahren, auf die in dieser Ausarbeitung nicht weiter eingegangen wird. Wichtig ist im laufenden Betrieb, dass möglichst schnell eine Lösung gefunden wird, auch wenn diese vielleicht nicht exakt der Optimallösung entspricht. Daher würden sich Heuristiken anbieten. In [24] wird zur Lösung des basierenden Problems ein dynamisches Programmierungsverfahren verwendet. Aus den Lösungen könnten Betriebsführungsstragien für Batterien und Erzeuger abgeleitet werden.

3.4 Auswirkungen und wirtschaftliche Aspekte

Die Maximierung des Eigenverbrauches, um geforderte Begrenzungen der Generatorleistung am Netzanschlusspunkt zu realisieren, kann auch zu bewussten Verbrauchserhöhungen führen. So könnte z.B. auf eine vergleichsweise ineffiziente, rein elektrische Warmwasseraufbereitung umgestellt werden, um die geforderten Quoten zu erreichen [13]. Gleichzeitig sinkt der Anreiz, verbrauchsintensive Geräte durch sparsame Geräte zu ersetzen. Sinnvoll wäre daher z.B. eine Koppelung der Förderung an die Entwicklung des Gesamtstromverbrauches. Alternativ wäre auch die Nutzung überschüssiger Energie für das Aufladen von Elektrofahrzeugen denkbar, um konventionelle Fahrzeuge und damit nicht erneuerbare Rohstoffe wie Erdöl zu substituieren [13].

Die dargestellten Rahmenbedingungen führen dazu, dass künftig vor allem kleinere PV-Anlagen installiert werden. Lag die Durchschnittsgröße neu in Deutschland installierter, geförderter PV-Anlagen im Feburar 2013 noch bei 25,21 kWp, ergab sich im Februar 2014 ein Mittelwert von lediglich 20,41 kWp [6]. Wie in Kapitel 3.3.2.3 dargestellt, sind bei Installation von Speichern positive Auswirkungen auf Lastspitzen und damit auf Investitionen in Stromnetze nur zu erwarten, wenn eine netzdienliche Betriebsführung verwendet wird. Für die Verwendung dieser Strategie leistet die Batteriespeicherrichtlinie durch die Begrenzung der Maximalleistung einen sinnvollen Beitrag.

Weiterhin ergibt sich die Frage, in wie weit eine Maximierung des Eigenverbrauches wirtschaftlich ist und ob sich der Einsatz von Speichertechnologie lohnt. Wie in Kapitel 3.2.2 dargestellt ist für neuinstallierte Anlagen der Eigenverbrauch i.d.R. wirtschaftlicher als eine Einspeisung ins Stromnetz. Sobald die Kosten einer gespeicherten Kilowattstunde geringer sind als die Differenz aus Strompreis und Einspeisevergütung, lohnt sich die Anschaffung eines Speichers zur Erhöhung des Eigenverbrauches [27]. Aktuell können Speicherkosten von weniger als 20 Ct/kWh erreicht werden [34], sodass dieser Zeitpunkt noch nicht erreicht ist. Die in Kapitel 3.2.3.1 dargestellte, geplante Besteuerung des Eigenverbrauches würde diesen

Zeitpunkt weiter in die Zukunft verschieben. Ein kostendeckender Betrieb kleinerer Speicher oder eines Heizstabes ist aber bereits möglich [17]. Bei größeren Anlagen reicht ein Speicher zur Maximierung des Eigenverbrauches i.d.R. nicht aus (vgl. Kapitel 3.3.2.2). Je mehr Speichertechnologie aber notwendig ist, desto stärker steigen die Kosten. Somit wird die Maximierung des Eigenverbrauchs teurer, je größer die temporären Differenzen aus Erzeugung und Verbrauch sind.

3.5 Fazit

Bei der Beschäftigung mit der Optimierung des Eigenverbrauches sollte zunächst eine Festlegung erfolgen, ob tatsächlich eine Maximierung des Eigenverbrauches oder ein hoher Autarkiegrad erreicht werden soll, da sich beide Ziele in der Regel nicht miteinander kombinieren lassen. Ferner sollte festgelegt werden, welcher Eigenverbrauchsgrad tatsächlich erreicht werden soll: Je höher der bereits erzielte Wert, desto teurer wird häufig es auch, diesen weiter zu erhöhen.

Es wurde dargestellt, dass bereits nutzbare Speicher in vielen Haushalten bereits vorhanden sind und damit technisch einfache sowie günstig realisierbare Lösungen, etwa ein Heizstab, den Eigenverbrauch stark erhöhen können. Bei Verwendung eines Stromspeichers ist eine netzdienliche Betriebsführung wichtig. Außerdem spielt die Speichergröße eine wichtige Rolle, um einerseits eine hohe Lebensdauer durch wenige Zyklen zu ermöglichen. Andererseits sollte die Kapazität innerhalb der definierten Grenzen auch ausgenutzt werden.

Weiteres Potential zur Optimierung des Eigenverbrauches besteht durch Nachfragemanagement. In Hotels könnte eine einfache Variante etwa dadurch realisiert werden, dass vorhandene Waschmaschinen mit Münzprüfer zur Mittagszeit günstiger benutzbar sind. Die Prozesse könnten ebenfalls dahingehend angepasst werden, dass durch das Hotelpersonal ausgeführte Waschgänge ebenfalls mittags ausgeführt werden müssen. Im gewerblichen Bereich spielen durch sich häufig wiederholende, ähnliche Tagesabläufe genaue, möglichst individuelle Lastprofile somit eine noch wichtigere Rolle.

Gleichzeitig wurde deutlich, dass neben Speichertechniken vor allem die Integration sämtlicher Erzeuger und Speicher wesentlich ist. Im Idealfall werden zusätzlich die Energieverbraucher integriert. Dafür ist eine Kommunikationsinfrastruktur, die auch entsprechende Prognosen berücksichtigt, hilfreich. Diese Prognosen über Verbrauch und Erzeugung müssen in einer hohen Datenqualität vorliegen und zeitlich möglichst hoch aufgelöst sein. Die Kommunikationsinfrastruktur berechnet z.B. mithilfe einer Steuereinheit ein optimales Verhalten, informiert daraufhin Erzeuger, Speicher und Verbraucher über relevante Daten, sendet ggfs. Steuersignale und erhält anschließend aktualisierte Werte. Die Berechnung des (möglichst) optimalen Verhaltens kann mithilfe mathematischer Optimierungsverfahren oder Heuristiken in zeitlich geringen Abständen erfolgen.

Literaturverzeichnis

1. EEG-Reform. Website des Bundesministeriums für Wirtschaft und Energie (BMWI). URL http://www.bmwi.de/DE/Themen/Energie/Erneuerbare-Energien/eeg-reform.html. Letzter Zugriff 09.05.2014
2. Energiewende - Anteil erneuerbarer Energien wächst weiter. Website des Presse- und Informationsamts der Bundesregierung. URL http://www.bundesregierung.de/Content/DE/Artikel/2014/01/2014-01-13-bdew-energiebilanz-2013.html. Letzter Zugriff 22.05.2014
3. Entwurf eines Gesetzes zur grundlegenden Reform des Erneuerbare-Energien-Gesetzes und zur Änderung weiterer Bestimmungen des Energiewirtschaftsrechts. Von der Bundesregierung am 08. April 2014 beschlossener Gesetzesentwurf
4. Gesetz für den Vorrang Erneuerbarer Energien (Erneuerbare-Energien-Gesetz - EEG). Gesetz vom 25. Oktober 2008 (BGBl. I S. 2074), das zuletzt durch Artikel 5 des Gesetzes vom 20. Dezember 2012 (BGBI. I S. 2730) geändert worden ist
5. Investitionen in Stromnetze notwendig für europäisches Netz erneuerbarer Energie. Website des Europäischen Parlamentes. URL http://www.europarl.europa.eu/news/de/news-room/content/20120203STO37180/html/Investitionen-in-Stromnetze-notwendig-f%C3%BCr-europ%C3%A4isches-Netz-erneuerbarer-Energie. Letzter Zugriff 28.05.2014
6. Photovoltaikanlagen: Datenmeldungen sowie EEG-Vergütungssätze - Monatliche Veröffentlichungen der PV-Meldezahlen. Webseite der Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahn. URL http://www.bundesnetzagentur.de/cln_1431/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/ErneuerbareEnergien/Photovoltaik/DatenMeldgn_EEG-VergSaetze/DatenMeldgn_EEG-VergSaetze_node.html. Letzter Zugriff 27.05.2014
7. Richtlinien zur Förderung von stationären und dezentralen Batteriespeichersystemen zur Nutzung in Verbindung mit Photovoltaikanlagen vom 21. Dezember 2012. Bekanntmachung des Bundesministeriums für Umwelt, Naturschutz und Reaktorsicherheit; veröffentlicht am Freitag, 19. April 2013, BAnz AT 19.04.2013 B1
8. Stromhandel. Webseite des Bundesministeriums für Wirtschaft und Energie. URL http://www.bmwi.de/DE/Themen/Energie/stromhandel.html. Letzter Zugriff 08.05.2014
9. Wechselrichter liefern die Basisdaten - Online-Dienst erlaubt präzise Solarprognosen. Website der EnBauSa GmbH, Online-Magazin für energetisches Bauen und Sanieren. URL http://www.enbausa.de/solar-geothermie/aktuelles/artikel/online-dienst-erlaubt-praezise-solarprognosen-1943.html. Letzter Zugriff 09.05.2014
10. Blanz, J., Rothert, M., Wachenfeld, V.: Technische und wirtschaftliche Aspekte der Zwischenspeicherung von Solarenergie zur Steigerung des Direktverbrauchs. Tech. rep., SMA Solar Technology AG (2010)
11. Bocklisch, T.: Intelligente dezentrale Energiespeichersysteme. uwf UmweltWirtschaftsForum **22**(1), 63–70 (2014). DOI 10.1007/s00550-013-0301-4. URL http://dx.doi.org/10.1007/s00550-013-0301-4. Springer Berlin Heidelberg
12. Bollinger-Kanne, J.: Speichertechnologien - Schöne smarte Speicherwelt. EW : das Magazin für die Energie-Wirtschaft. Frankfurt, M : EW-Medien u. Kongresse, ISSN 0013-5496, ZDB-ID 20714440. - Vol. 112.2013, 12, p. 40-41 (2013)
13. Bost, M., Hirschl, B., Aretz, A.: Effekte von Eigenverbrauch und Netzparität bei Photovoltaik - Langfassung. Tech. rep., Studie im Auftrag von Greenpeace Energy eG für das Institut für ökologische Wirtschaftsforschung (IÖW) (2011)
14. Enkhardt, S.: Neuer EEG-Entwurf: 50 Prozent EEG-Umlage auf Photovoltaik-Eigenverbrauch. Website pv magazine Deutschland der Solarpraxis AG. URL http://www.pv"=magazine.de/nachrichten/details/beitrag/neuer"=

eeg-entwurf"="50"=prozent"=eeg"=umlage"=auf"=photovoltaik"=eigenverbrauch_100014824/. Letzter Zugriff 09.05.2014
15. Frielingsdorf, J.: Ergebnisse der Erhebung: Wo im Haushalt bleibt der Strom? Website der EnergieAgentur.NRW GmbH. URL http://www.energieagentur.nrw.de/presse/singles-verbrauchen-strom-anders-15327.asp. Letzter Zugriff 28.05.2014
16. Gerblinger, A., Wiest, M.: Erzeugung - Große Anlage ohne Speicher am wirtschaftlichsten - Eigenverbrauch in Privathaushalten — Chance mit vielen Facetten. EW : das Magazin für die Energie-Wirtschaft. Frankfurt, M : EW-Medien u. Kongresse, ISSN 0013-5496, ZDB-ID 20714440. - Vol. 112.2013, 12, p. 42-45 (2013)
17. Huber, M.: ERNEUERBARE ENERGIEN - Das "Post-EEG"-Potenzial von Photovoltaik im privaten Strom- und Wärmesektor. Energiewirtschaftliche Tagesfragen : et; Zeitschrift für Energiewirtschaft, Recht, Technik und Umwelt. - Essen : Etv GmbH, ISSN 0013-743X, ZDB-ID 2041443. - Vol. 63.2013, 9, p. 57-61 (2013)
18. Jung, S.: Förderung des Eigenverbrauch von Solarstrom - Rechtliche Informationen und deren praktische Umsetzung. Website des Solarenergie-Förderverein Deutschland e.V. (SFV). URL http://www.sfv.de/artikel/2008/foerderung_des_eigenverbrauchs_von_solarstrom.htm. Letzter Zugriff 09.05.2014
19. Kondziella, H., Brod, K., Bruckner, T., Olbert, S., Mes, F.: Stromspeicher für die „Energiewende" – eine akteursbasierte Analyse der zusätzlichen Speicherkosten. Zeitschrift für Energiewirtschaft **37**(4), 249–260 (2013). DOI 10.1007/s12398-013-0115-7. URL http://dx.doi.org/10.1007/s12398-013-0115-7. Springer Fachmedien Wiesbaden
20. Kost, C., Mayer, J.N., Thomsen, J., Hartmann, N., Senkpiel, C., Philipps, S., Nold, S., Lude, S., Schlegl, T.: Stromgestehungskosten Erneuerbare Energien. Frauenhofer-Institut für solare Energiesysteme ISE (2013)
21. Märtel, C.: Eigenverbrauch von Solarstrom. Website des DAA Deutsche Auftragsagentur GmbH. URL http://www.photovoltaik-web.de/eigenverbrauch-pv.html. Letzter Zugriff 09.05.2014
22. Paul, D.: Bidirektionale Datenkommunikation -EEG und Rundsteuertechnik: Passt das zusammen? EW : das Magazin für die Energie-Wirtschaft. Frankfurt, M : EW-Medien u. Kongresse, ISSN 0013-5496, ZDB-ID 20714440. - Vol. 112.2013, 12, p. 28-29 (2013)
23. Quaschning, V.: Regenerative Energiesysteme, 8 edn. Carl Hanser Verlag GmbH und Co. KG; 424 p. (2013)
24. Riffonneau, Y., Bacha, S., Barruel, F., Ploix, S.: Optimal Power Flow Management for Grid Connected PV Systems With Batteries. Sustainable Energy, IEEE Transactions on **2**(3), 309–320 (2011). DOI 10.1109/TSTE.2011.2114901
25. Sonnenschein, M., Rapp, B., Bremer, J.: Demand Side Management und Demand Response. Handbuch Energiemanagement; Beitrag Nr. 10620; EW Medien und Kongresse GmbH (2010)
26. Stahl, L.F.: Einbindung eines BHKW - Kombinierter Eigenverbrauch von BHKW mit PV. Website des BHKW-Forum e.V.; Redaktion BHWK-Infothek. URL http://www.bhkw-infothek.de/bhkw-informationen/technische-grundlagen/einbindung-eines-bhkw/. Letzter Zugriff 14.05.2014
27. Struth, J., Leuthold, M., Aretz, A., Bost, M., Gährs, S., Cramer, M., Szczechowicz, E., Hirschl, B., Schnettler, A., Sauer, D.U.: Thesen und Hintergründe zum Nutzen von Speichern in netzgekoppelten PV-Anlagen. Bundesministerium für Umwelt, Naturschutz und Reaktorsicherheit (BMU) (November 2013)
28. Tamme, R.: Speichertechnologien für erneuerbare Energien - Voraussetzung für eine nachhaltige Energieversorgung. In: FVS Themenheft 2006, p. 82-90. Oktoberdruck AG. FVS Jahrestagung 2006, 2006-09-21 - 2006-09-22, Berlin. ISSN 0939-7582 (2006)
29. Weniger, J., Bergner, J., Tjaden, T., Quaschning, V.: Bedeutung von prognosebasierten Betriebsstrategien für die Netzintegration von PV-Speichersystemen. In: 29. Symposium Photovoltaische Solarenergie, Kloster Banz, Bad Staffelstein (12.-14.03.2014)
30. Wiest, M., Finkel, M., Engel, B.: Innovatives Energiemanagement bei Haushaltskunden – ein Beitrag zur Netzstabilität? In: 13. Symposium Energieinnovation: Innehalten und Ausblick - Effektivität und Effizienz für die Energiewende, Graz, Österreich (12.-14.02.2014)

31. Winkels, L., Schmedes, T., Appelrath, H.J.: Dezentrale Energiemanagementsysteme. Wirtschaftsinformatik; Vieweg Verlag / Wiesbaden **49**(5), 386–390 (2007)
32. Wirth, D.H.: Aktuelle Fakten zur Photovoltaik in Deutschland. Tech. rep., Fraunhofer-Institut für Solare Energiesysteme ISE (10.04.2014)
33. Wittwer, C.: Dezentrale PV-Batteriespeicher erweitern Spielraum - Mehr Strom aus Photovoltaik ohne Netzausbau. EW : das Magazin für die Energie-Wirtschaft. Frankfurt, M : EW-Medien u. Kongresse, ISSN 0013-5496, ZDB-ID 20714440. - Vol. 112.2013, 11, p. 74-76 (2013)
34. Wittwer, C.: Speicherstudie 2013 - Kurzgutachten zur Abschätzung und Einordnung energiewirtschaftlicher, ökonomischer und anderer Effekte bei Förderung von objektgebundenen elektrochemischen Speichern - Zusammenfassung der wichtigsten Erkenntnisse. Frauenhofer-Institut für Solare Energiesysteme ISE in Freiburg (2013)
35. World Bank (Hrsg.): Primer on Demand-Side Management: With an Emphasis on Price-Responsive Programs. World Bank Other Operational Studies 8252, The World Bank (2005). URL http://ideas.repec.org/p/wbk/wboper/8252.html

Kapitel 4
Gebäude-Energiemanagementsysteme

Dag Ennenga

4.1 Einleitung

Seit nun bereits einigen Jahren ist die Zeit des billigen Stroms endgültig vorbei. Stetig steigende Kosten für Energie auf Grund von Ressourcenknappheit bei den Primärenergieträgern Öl und Gas, sowie der geplante Atomausstieg in Deutschland bringen viele Veränderungen für das Stromnetz mit sich. Die Erzeugung von Energie ist immer mit Verlusten verbunden, was eine möglichst effiziente Nutzung der verbleibenden Ressourcen interessanter macht. Das Umwandeln von elektrischer in andere Energien ist einfach möglich. Dennoch ist eine längerfristige Speicherung von Energie ökonomisch nicht sinnvoll, da bisher keine Art von Speicher mit ausreichendem Wirkungsgrad bekannt ist. Das macht die zeitnahe Verwendung von erzeugtem Strom notwendig. Um eine ineffiziente Speicherung von Energie zu vermeiden ist also im gesamten Stromnetz ein Anpassen von Erzeuger und Verbraucher die einzige Lösung.
Zur Anpassung von Erzeugung und Verbrauch wird sogenannte Regelleistung genutzt, welche sowohl bei Überangebot als auch bei Unterversorgung von Energie ein Gleichgewicht schaffen soll. Das Problem der Regelleistung ist, dass ihre Bereitstellung höhere Kosten als die Stromerzeugung verursacht.
Hinzu kommt, dass die Erzeugung von Strom aus Erneuerbaren Energien (EE) in Deutschland immer wichtiger wird. Nicht nur auf Grund von Ressourcenknappheit, sondern auch in Hinsicht auf die Reduzierung der Treibhausgasemission, insbesondere der CO2-Ausstoß. Bei der Brutto-Stromerzeugung 2013 lag der Anteil erzeugter Energie von EE bereits bei 23,4% [1]. Die Photovoltaik und Windenergie machen dabei gemeinsam 12,4% der Gesamterzeugung aus. Doch gerade die fluktuierende Einspeisung gewonnener Energie aus diesen Technologien bringt einen zusätzlichen Regelenergiebedarf mit sich.

Carl von Ossietzky Universität Oldenburg
E-mail: dag.ennenga@uni-oldenburg.de

Seit einigen Jahren wird bereits nach besseren Strategien gesucht um durch automatisiertes Lastenmanagement, bessere Messmethoden und Prognosemodelle einen sparenden Kostenfaktor zu generieren. Die externe Steuerung von größeren Energieverbrauchern durch die Energieversorgungsunternehmen (EVU), um ein mögliches Ungleichgewicht im Stromnetz abzufangen, wird bereits angewandt. Durch Flexible Stromtarife im Verbund mit Preissignalen soll kleineren Verbrauchern ein Bewusstsein für die Thematik vermittelt werden. Mit einem flexiblen Einsatz von Verbrauchern kann ebenfalls kurzfristig eine Veränderung im Lastverhalten erzeugt und der Nutzen möglicher eigener Erzeuger kann optimiert werden.

4.2 Gliederung

Begonnen wird die Ausarbeitung mit einer Motivation, die deutlich macht, wie wichtig das Thema auf Grund des Wandels im Stromnetz und der EE ist. Darauf folgt eine Einführung in unser heutiges Stromnetz, die als Grundlage notwendig ist, insbesondere die Regelenergie wird behandelt. Über die Thematik des Energiemanagements wird der Hauptteil, das Gebäudeenergiemanagement eingeleitet. Nach einem Beispiel für den Einsatz von Gebäude-Energiemanagementsystemen wird auf die klassischen Lastmanagementverfahren eingegangen, die in dieser Form auch verwendet werden. Ein Bezug zur Projektgruppe “Hardwarebasierte Simulation energieautonomer Gebäude” wird im Folgekapitel hergestellt. Abschließend ist eine kurze Zusammenfassung enthalten.

4.3 Motivation

Ein Leben ohne Verwendung von Elektrizität ist für den Großteil unserer Bevölkerung nicht mehr vorstellbar. Alleine die Gebäude in Deutschland machen beim Gesamtenergieverbrauch einen Anteil von ca. 40% aus. Dabei werden die meisten Geräte in Gebäuden mit Elektrizität betrieben. Durch energieeffizientere Gebäude kann bereits ein wesentlicher Beitrag zu den Klimaschutzzielen in der EU beigetragen werden.
Die Energienachfrage wird weltweit weiterhin ansteigen, wobei der Großteil fossiler Energieträger sich in politisch instabilen Regionen befindet. Hinzu kommt, dass die bereits gesicherten und wirtschaftlich ertragreichen Vorräte in absehbarer Zeit aufgebraucht sind [3]. Ohne eine möglichst effiziente Nutzung erneuerbarer Energie wird also nicht nur der Strompreis weiterhin stark ansteigen, sondern ein Defizit bei den Energieerzeugern entstehen, da in Deutschland gleichzeitig der Ausstieg aus der Atomenergie geplant ist.
Für die Stabilität im Stromnetz findet vermehrt eine Anpassung des Angebots an der Nachfrage von Energie statt. Dieses System setzt allerdings die Bereitstellung einer großen Menge an positiver Regelenergie voraus. Regelenergie, die zusätzlich erzeugt

werden muss um ungeplant hohen Verbrauch auszubalancieren. Die Kraftwerke zur Generierung von Regelenergie werden nur zu Spitzenlastzeiten eingesetzt und erzeugen sonst keine Energie. Sie stellen einen hohen Kostenfaktor dar, da eine Anlage ohne Erzeugung keine Einnahmen dennoch laufende Kosten generiert. Durch eine Anpassung von der Nachfrage an das Angebot, wird nicht nur eine geringere Umweltbelastung erzeugt, sondern gleichzeitig eine Kostenoptimierung für den Kunden, durch weniger benötigte positive Regelenergie, erreicht. Statt dessen wird negative Regelenergie notwendig. Das bedeutet Verbraucher müssen notfalls hinzu geschaltet werden um den Verbrauch anzupassen. Zusätzlich kann von großen Energieverbrauchern Regelenergie für den Strommarkt zur Verfügung gestellt werden, wenn ein entsprechendes Management vorhanden ist.
Energieintensiven Unternehmen steht außerdem eine Steuerbefreiung bei Einhaltung des EEG zu, dessen EEG-Umlage momentan bei 6,240 ct/kWh liegt. Für jegliche Unternehmen ist ein positives Rating bei Geschäftspartner und Geldgebern allerdings wichtig. Eine hohe Energieeffizienz und hohes Umweltbewusstsein leistet dabei einen wichtigen Beitrag.
Um die angesprochenen Ziele erreichen zu können ist eine möglichst sinnvolle Abstimmung von Energieverbrauchern, -erzeugern und auch Energiespeichern notwendig. Mit einem passenden Energiemanagement können beispielsweise in Krankenhäusern ein Einsparpotenzial von bis zu 30% ausgeschöpft werden [9].

4.4 Unser Stromnetz

In Deutschland sorgen vier Übertragungsnetzbetreiber (ÜNB), zusammengeschlossen zum deutschen Netzregelverbund, für die Stromversorgung durch die stromerzeugenden Kraftwerke der EVU. Die ÜNB sind verantwortlich für den Betrieb, den Ausbau und die Wartung der Transportnetze und die Netzsicherheit und Stromqualität werden von Ihnen garantiert. Die sich addierende Abweichung zwischen Stromerzeugung und Stromverbrauch muss auf Höchstspannungsebene ausgeglichen werden. Diese sogenannte Regelenergie wird von den ÜNB als Dienstleistung bereitgestellt werden [6].
Die Höchstspannungsebene (220kV oder 380kV) dient als überregionales Transportnetz zur Verbindung der nationalen Stromnetze. Über Transformatoren wird der Strom in andere Spannungsebenen eingespeist. Das Hochspannungsnetz (50kV-150kV) übernimmt dabei die Rolle des regionalen Verteilnetzes zur Verbindung von Ballungszentren. Über etwa 900 weitere Verteilnetzbetreiber gelangt der erzeugte Strom zu den Endverbrauchern. Sie betreiben die Mittelspannungs- (6kV-30kV) und Niedrigspannungsebene (230V-680V). Mit der Mittelspannungsebene wird innerhalb von Ballungsräumen der Strom verteilt oder verschiedene Industrie- und Gewerbekunden sind, zur effizienten Versorgung mit Energie, direkt an dieses Netz angeschlossen. Diese Eben wird ebenfalls genutzt um mittelgroße Stromerzeugungsanlagen wie etwa städtische BHKW-Anlagen in das Stromnetz einzuspeisen. Mit dem Niedrigspannungsnetz sind die übrigen Kunden mit kleinen Geräten und

Anlagen verbunden, sowie beispielweise Haushalte. Das gesamte Netz wird auch Verbundnetz genannt. Auf das Gegenstück von Inselnetzen wird hier nicht weiter eingegangen.
Beim Gesamtstrombedarf in Deutschland wurde im Jahr 2013 ein Anteil von 23,4% aus EE gewonnen. Durch fossile Rohstoffe gewonnene Energie hat einen Anteil von etwa 56% (hauptsächlich Kohle), 15% etwa werden durch Kernenergie produziert. Im Gegensatz zum Jahr 2005 ist der Anteil an Kernenergie (26%) und Energie aus fossilen Rohstoffen (60%) zurückgegangen.

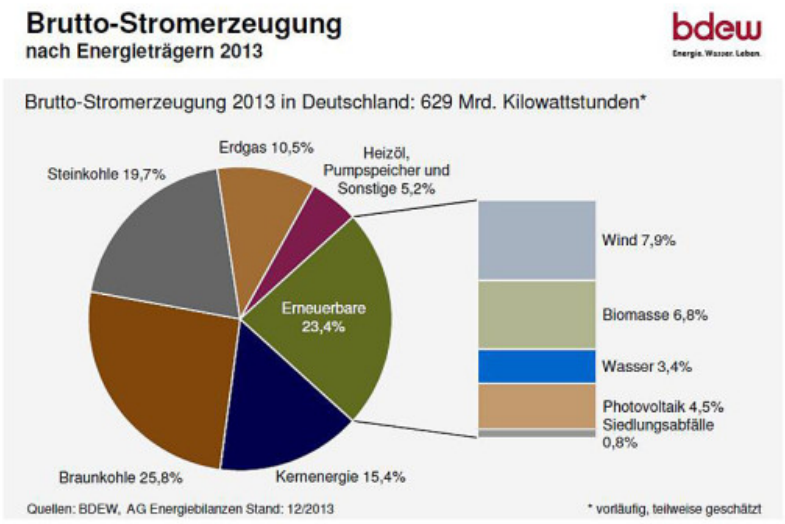


Abb. 4.1 Brutto-Stromerzeugung nach Energieträgern 2013 [1]

Diese Veränderung ist auf das Erneuerbare-Energie-Gesetz, welches die Bedingungen für Kraftwerke mit EE deutlich verbessert, und auf den geplanten Atomausstieg Deutschlands zurückzuführen. Ebenfalls ist ein Wachstum an Kraft-Wärme-Kopplung-Anlagen (KWK-Anlagen) in Deutschland bis 2020 geplant. Die Folge aus diesen Veränderungen ist eine zunehmende Dezentralisierung bei der Stromerzeugung, welche neue Anforderungen an den Betrieb der Stromnetze stellt. Insbesondere muss die Veränderung bei der Bereitstellung von Regelenergie berücksichtigt werden.

4.4.1 Regelenergie

Obwohl das Stromnetz allgemein hohen Schwankungen, insbesondere auch durch fluktuierende EE, ausgesetzt ist, fällt die Stromversorgung so gut wie nie aus. Einerseits liegt dies an einem möglichst genauen Prognosen für die Liefermengen auf Seite der Stromproduzenten, wodurch die Einspeisung ins Stromnetz optimal geplant werden kann. Andererseits kann auf plötzlich ansteigenden oder sinkenden Stromverbrauch, der in der Prognose nicht berücksichtigt werden konnte, mit Regelenergie ein Zusammenbruch des Stromnetzes verhindert werden. Die innerhalb von Sekunden (Primärreserve), 5 Minuten (Sekundärreserve) oder einer Viertelstunde (Minutenreserve) erzeugte Energie sorgt dafür, dass das Stromnetz ausgeglichen ist [7].

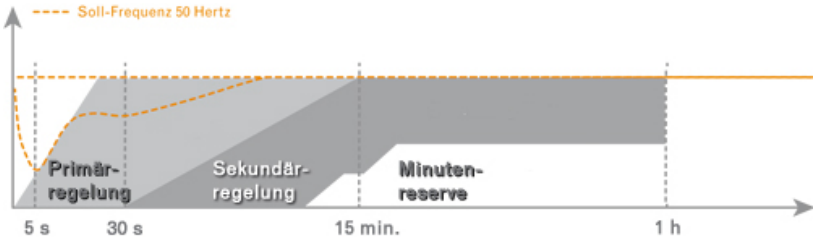


Abb. 4.2 Einspeisung von Regelenergie, Primär-, Sekundär-, Minutenreserve [2]

Die sogenannte Stundenreserve ist nicht mehr im Begriff der Regelenergie enthalten. Nach 60 Minuten muss die Minutenreserve durch die Stundenreserve abgelöst und vom Verantwortlichen für die Stromschwankung im Netz gestellt werden. Dies kann durch Regelung eigener Kraftwerke oder Zu-/Verkauf von zusätzlicher Energie passieren.
Regelenergie kann allgemein in zwei Fällen von Nöten sein. Bei plötzlich ansteigendem Verbrauch von Strom spricht man von *positiver Regelenergie*, die erzeugt und ins Stromnetz eingespeist werden muss. Es kann aber genau so der gegenteilige Fall eintreten, bei dem ein erhöhtes Angebot an Strom vorhanden ist, auf Grund von plötzlichem Absinken der Stromnachfrage. In diesem Fall ist eine Speicherung des Stroms bzw. das Herunterregeln von Kraftwerken notwendig und man spricht von *negativer Regelenergie* [7].
Die benötigte Regelenergie zwischen 4.500 und 6.000 Megawatt (jeweils für positive und negative Regelenergie) wird von den ÜNB bereitgestellt. Vergütet werden die Anlagenbetreiber über einen Leistungspreis. Beim seltenen Einsatz der Anlagen erhalten sie zusätzlich einen Arbeitspreis. Trotz hohen Schwankungen für den Arbeitspreis auf dem Regelenergiemarkt, sind die Preise im Allgemeinen um ein Vielfaches höher als der Normalstrompreis und damit vorteilhaft für die Produzenten von Regelenergie.

Der Stromverbraucher zahlt über die Netznutzungsentgelte dabei den Leistungspreis, also die Vorhaltung der Regelenergiekapazität und erhalte dafür eine unterbrechungsfreie Stromversorgung. Der Arbeitspreis wird über Ausgleichsenergie verrechnet. Mit dem Begriff Ausgleichsenergie wird die Umlage der Abrufkosten der Regelenergie auf die verschiedenen Akteure im Stromnetz bezeichnet. Während die Regelenergie das tatsächliche physische Gleichgewicht des Stromnetzes gewährleistet, sorgt die Ausgleichsenergie für das bilanzielle Gleichgewicht. Wird der geplante Fahrplan von Erzeuger oder Verbraucher nicht eingehalten, muss Regelenergie erzeugt werden. Diese Regelenergie wird dem Verursacher mit Ausgleichsenergie in Rechnung gestellt. Sonderfall ist wenn sowohl Verbraucher und Erzeuger gleichmäßig weniger oder mehr Energie erzeugen und verbrauchen, dann kommt es auch zu keiner Erzeugung von Regelenergie.

4.5 Energiemanagement

Das Ziel von Energiemanagement ist es Unternehmensziele (oder allg. Ziele) bei minimalen Energiekosten zur erreichen. Im Krankenhaus wäre das zum Beispiel eine gute Versorgung für alle Patienten und dafür einen funktionalen Arbeitsplatz für Angestellte der Klinik zu gewährleisten. Diese minimalen Energiekosten erreicht man generell durch zwei Methoden. Erstens, indem man *weniger bezahlt pro verbrauchter Energieeinheit* oder zweitens, durch *Reduzierung des Energieverbrauchs*. Um weniger pro Energieeinheit zu bezahlen gibt es wiederrum unterschiedliche Möglichkeiten.

- Findung eines bestmöglichen Tarifs für seinen Stromverbrauch, durch Ermitteln der Verbraucher und Erstellen eines Lastprofils
- Kosten vermeiden: Durch Lastabwurf um Spitzenpreise zu vermeiden, Verbesserung der Auslastung und Verringerung der maximalen Nachfrage
- Preisorientiertes Anpassen der verwendeten Ressource bei beispielsweise Heizkesseln, die sowohl Öl als auch Gas zum Heizen verwenden können.
- On-Site Anlagen: Verwendung von KWK-Anlagen insbesondere für Verbraucher die eine ganzjährige Wärmelast haben (z.B. Krankenhäuser), oder das Testen von Notstromaggregaten zu Höchstpreiszeiten.

Die Reduzierung des Gesamtverbrauchs innerhalb von Gebäuden ist beim Neubau natürlich einfacher und kostengünstiger zu verwirklichen. In bereits vorhandenen Gebäuden ist zum Beispiel ein Energieaudit eine Möglichkeit Einsparpotentiale zu entdecken. Diese Einsparpotentiale können bei einer nachträglichen Sanierung oder Anschaffung zusätzlicher Anlagen (oft miteinander verbunden) ausgeschöpft werden. Eine andere Möglichkeit bietet die optimierte Verwendung von bereits installierten Anlagen, die den Gesamtverbrauch senkt. Dies geschieht wieder durch zwei Ansätze. Vermeiden von Energieverschwendung ist ein proaktiver Ansatz. Zum Beispiel kann Energieumwandlung (Heizkessel), Energieverteilung (Rohrdämmung) oder der mög-

lichst effiziente Betrieb von Maschinen in der Produktion und deren Instandhaltung können Energieverschwendung vermeiden [10].
Ein reaktiver Ansatz dagegen ist das Erkennen von Energieverschwendung durch ein Monitoring System. Wie gut ein Gebäude auch betrieben wird, es kommt immer zu Fehlern, technischen Problemen, oder fehlerhafter Nutzung, die zu Energieverlusten führt. Ein gutes Monitoring System erkennt diese Fehler und trägt dazu bei nachträgliche Energieverschwendung zu verhindern.
Ein gelungenes Energiemanagement wird innerhalb eines Unternehmens von drei Bereichen getragen. Der Bereich der Organisation und Information wird vom Management übernommen, die sich für eine Energiepolitik und -strategie entscheiden und finanzielle Mittel für die Installation von Systemen zum Energieeinkauf, Datensammlung, Analyse und Berichterstattung zur Verfügung stellen. Der zweite Bereich sind alle weiteren Nutzer des Gebäudes die mit den Anlagen in Berührung gelangen. Sie benötigen die Motivation und das Wissen die Anlagen entsprechend zu bedienen. Und der dritte Bereich bildet die Technik. Dieser kann Lüftung, Klimaanlage, Heizung, Warmwasser, Licht, Maschinen, Computerräume, Aufzüge, Solaranlagen, Energiespeicher, Kontrollen, Gebäude-Energiemanagementsysteme und weiteres enthalten.
Ein Gebäudeenergiemanagementsystem kann beispielsweise nur effizient eingesetzt werden, wenn der Nutzer auch entsprechend geschult wurde. Eine ausgewogene Energiestrategie ist also notwendig um Nutzer und Technik miteinander zu verbinden.

4.5.1 Gebäudenergiemanagement

Ein Gebäude-Energiemanagementsystem (engl. Building Energy Management System
(BEMS)) ist eine technische Lösung für die Problematiken aus dem Bereich des Energiemanagements. Sie sind innerhalb der Energiemanagementsysteme (EMS) der wichtigste und größte Bereich. Mit dem System sollen möglichst alle Energiesparmöglichkeiten wahrgenommen werden. Eine kontinuierliche Überwachung der Energieverbraucher soll ermöglicht werden, wobei die Nutzungsqualität aller Anlagen gleichbleibend sein soll [10].

Definition: „Ein BEMS ist ein Computerbasiertes System das den Bereich der energierelevanten Betriebssysteme (Verbraucher / Erzeuger / Speicher) beobachtet und kontrolliert.“

Unter Verbrauchern versteht man bei dieser jegliche Art von Nutzern elektrischer Energie, insbesondere Klimaanlage, Lüftung, Heizung und Beleuchtung, die dauerhaft im Betrieb sind. Die Erzeuger können Photovoltaik-, Photothermie-Anlagen, Wärmepumpen, BHKW, etc. sein. Als Speicher sowohl thermische als auch elektrische Speicher oder Kombinationen denkbar. Nicht erwähnt, aber dennoch ebenfalls

in einem BEMS enthalten sein können Sicherheitssysteme, Zugangskontrollen, Feueralarmsysteme und andere Systeme.
Das BEMS kann dabei automatisches ein- und ausschalten von Anlagen veranlassen. Beispielsweise das Ausschalten von Licht bei ausreichend Tageslicht. Einige Anlagen können bei ihrer Ausführung optimiert werden. Eine Heizungsanalage könnte entsprechend der gegebenen Außentemperatur automatisch mit Start- und Stoppzeiten gesteuert werden. Gebäudemanager hätten durch ein Monitoring für den Anlagenstatus die Möglichkeit nicht regulierbare Anlagen in gegebener Zeit zu regulieren. Allgemeine Energiemanagementinformationen können durch das System bereitgestellt werden, wodurch wiederum schnell neue Entscheidungen getroffen werden können um die Energieeffizienz im Gebäude zu steigern. Ferngesteuert Überwachungs- und Kontrollmöglichkeiten der Anlagen führen außerdem zu einer effizienteren Auslastung der Arbeitskräfte. Eine ferngesteuerte Überwachung und Kontrolle über Telefonnetzwerk macht sogar ein Energiemanagement durch andere Dienstleister möglich. Verwaltungs- und Wartungspläne könnten ebenfalls in das BEMS integriert werden.
Die Hauptkomponenten eines kompletten BEMS sind dabei die Hardware, die Software, das Kommunikationsnetzwerk und gegebenenfalls benötigte Schnittstellen zu anderen Systemen. Hardwarekomponenten sind Großteils Außenstationen, die Verbraucher und Erzeuger kontrollieren und schalten, sowie Sensoren (z.B. Außentemperatur) und Aktuatoren die Befehle des BEMS ausführen. Eine Außenstationen kann dabei folgende Bestandteile oder Module umfassen: Inputs, Outputs, Mikroprozessor, Speicher, RAM, EPROM mit konfigurierbarer Strategie, Module, Zeit-Uhr, Netzteil, RS232-Port und mehr. Als Input können dabei Sensoren, Relais oder Energiemeter dienen. Der Mikroprozessor führt dabei die Steuervorgänge aus. Die Strategiekonfiguration ist auf dem Hauptrechner programmiert und wird von den Außenstationen heruntergeladen. Für diese Strategiekonfigurationen wird oftmals ein Lastmanagement angewendet.

4.5.2 Beispiel eines BEMS

Für das Beispiel eines BEMS soll das System von Cylon Controls Ltd. vorgestellt werden [5]. Cylon arbeitet seit 25 Jahren im Bereich des Energiemanagements und hat seinen Hauptsitz in Dublin, Irland. Bei dem von Cylon entwickelten System handelt es sich nicht um ein alleinstehendes Gebäude-Energiemanagementsystem. Statt dessen ist ihr hauseigenes Gebäudemanagementsystem um das Energieüberwachungssystem *Cylon Active Energy SaaS (CAE)* erweiterbar.

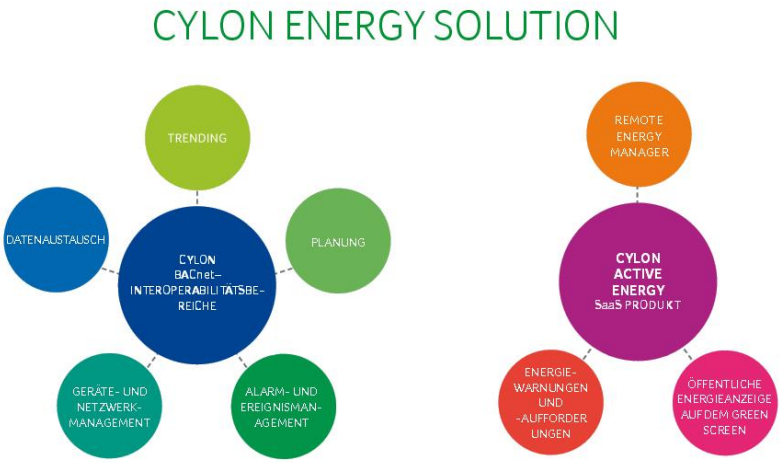


Abb. 4.3 Cylon Energy Solution [5]

Laut Cylon sind 90% aller BMS nicht energieeffizient, da ihre Informationen nicht analysiert und überwacht werden, sondern erzeugen zusätzliche Energiekosten. Genau dort soll Cylon Active Energy ansetzen und arbeitet mit dem BMS über BACnet oder SQL zusammen.BACnet ist ein von Cylon eigens entwickelte Lösung für die Integration der Gebäudeleittechnik. Das BACnet ermöglicht die Datenübertragung zwischen Geräten und erzeugt Alarm- und Warnmeldungen. Entsprechend dem Kunden und dessen Geräten ist das BACnet anpassbar. Das CAE liefert Echtzeitinformationen (alle 15 Minuten) über die Energienutzung und -verbrauch in einem Gebäude. Dafür enthält es Berichtsfunktionen wie Diagramme, Analysemöglichkeiten und Warnmeldungen über Benutzerschnittstellen. Dabei ist das CAE in 3 Bestandteile geteilt. Der erste Bestandteil wird für die Formalisierung von Energiewarnungen und Anforderungen an das System genutzt. Der Remote Energy Manager soll diese im Gebäude umsetzen und ist die zentrale Einheit zur Optimierung. Als dritter Bestandteil ist ein öffentliche Energieanzeige für den sogenannten Green Screen enthalten, diese Anzeige soll eine positive Verhaltensänderung zum Energiemanage-

ment bei Mitarbeitern erzeugen. Mit Tipps und Empfehlungen zur Reduzierung des Energieverbrauchs soll jeder Mitarbeiter mit einbezogen werden.
Das CAE ist ein Cloud-basierter Service, der mehreren Benutzern mit unterschiedliche Zugriffsoptionen bietet und über jeglichen online Zugriff zu erreichen ist.
Mit CAE werden für Gebäude eine Senkung von bis zu 25% beim Energieverbrauch erreicht und eine Kapitalrendite soll bereits nach spätestens 2 Jahren erreicht sein. Es unterstützt die Zertifizierung und Einhaltung der EN16001 und ISO 50001 Energiemanagement Normen zur Entwicklung von Energiemanagementsystemen.

4.5.3 Klassisches Lastenmanagement

Die Ziele des klassischen Lastmanagements sind die Verminderung von Lastspitzen, die Verschiebung des Lastprofils in günstigere Tarifzeitzonen (z.B. beim Nachtstromtarif), wodurch eine Kostenoptimierung für den Kunden entsteht. Gleichzeitig soll eine Anpassung des Energieverbrauchs an das Energieangebot geschaffen werden, insbesondere bei Verwendung eigener Energieerzeuger (z.B. Photovoltaikanlagen oder BHKW) um eine möglichst hohe Energieeffizienz zu erreichen. Das EVU sieht ebenfalls einen Vorteil im Lastmanagement und zwar führt eine gleichmäßige Last zu weniger unvorhergesehenen Lastspitzen und der Kraftwerksfahrplan kann besser eingehalten werden.
Das klassische Lastmanagement lässt sich in 4 Strategien unterteilen die im Folgenden näher erläutert werden sollen.

4.5.3.1 Peak Clipping

Das Peak Clipping wird zur Reduzierung der Spitzenlast eingesetzt. Dafür werden nicht unbedingt benötigte Verbraucher abgeschaltet, wodurch eine Minderung beim Gesamtenerägiebedarf erreicht wird. Dies kann beispielsweise durch Einsatz eines Maximumwächters erreicht werden.
Dafür wird eine Spitzenlast festgelegt, die im Tagesverlauf nicht überschritten werden soll. Beim Erreichen der Spitzenlast sollen gegebenenfalls Verbraucher abgeschaltet werden. Dafür vergleicht der Maximumwächter den IST- und SOLL-Wert eines 1/4-stündigen Mittelwertes. Für die Abschaltung der Anlagen wird den Anlagen einen Priorität zugewiesen, nach der sie in Reihe abgeschaltet werden. In umgekehrter Folge werden sie bei Rückgang der Last wieder zugeschaltet. Durch einen Maximumwächter werden sowohl die Leistungskosten als auch Leistungsbereitstellungskosten gesenkt. Die Frage nach der maximalen Spitzenlast ist beim Peak Clipping schwer zu beantworten, da sie zu groß gewählt einen geringen Einsparungsfaktor hat und zu niedrig gewählt Einbußen in der Qualität nach sich ziehen kann beim Betrieb innerhalb des Gebäudes. Vorherige Messungen sind also notwendig.

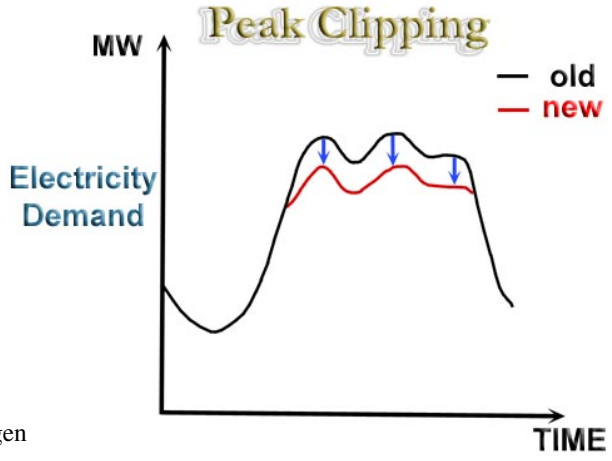


Abb. 4.4 Peak Clipping

Peak Clipping kann ebenfalls seitens der EVU angewandt werden, wobei via Rundsteuerung Verbraucher beim Kunden abgeschaltet werden können. Hierfür sind vertragliche Randbedingungen, wie Häufigkeit und Dauer des Lastabwurfs, sowie vertragliche Vergünstigungen für den Kunden festzulegen [8].

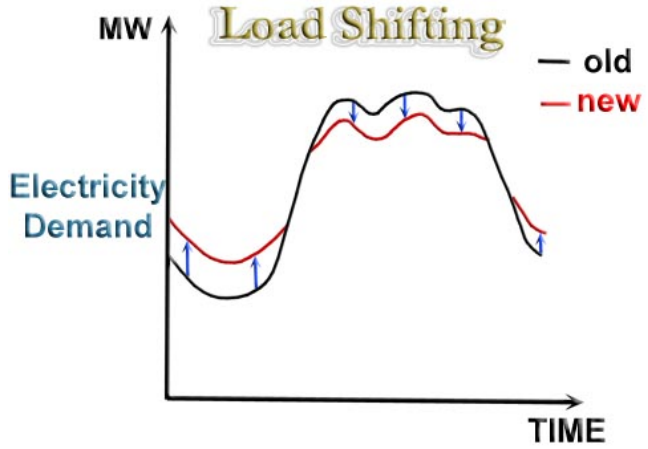
4.5.3.2 Valley Filling

Beim Valley Filling werden die vorhanden Lasttalzeiten mit Lasten aufgefüllt. Dies kann durch Umwandlung von elektrischer in thermische Energie geschehen. Bei einer festgelegten Minimallast sollen dabei die thermischen Speicher in Betrieb genommen und geladen werden. Somit geladene Speicher könnten als Antwort auf Preissignale (Demand Response) genutzt werden. Der Vorteil des Valley Filling ist

die Ausnutzung günstiger Nachttarife, da dort standardgemäß das größte Lasttal liegt.

4.5.3.3 Load Shifting

Durch Load Shifting werden Lasten von Spitzenlast- in Tallastzeiten verschoben. Für den Gesamtverbrauch des Gebäudes stellt das Load Shifting dabei keine Veränderung dar. Beispielsweise typische Geräte wie Waschmaschinen, Trockner oder



9cm

Abb. 4.5 Load Shifting

Geschirrspüler könnten möglichst zu Lasttalzeiten verwendet werden. Ebenfalls können Nachtspeicherheizungen oder andere Speicher, die nachts aufgeladen werden, für eine Verschiebung der Last verwendet werden. Kühlgeräte besitzen ebenfalls eine Speicherkapazität und könnten für das gesamte deutsche Stromnetz beim gleichzeitigen Ausschalten eine positive Regelenergie von 1,5 GW.

4.5.3.4 Load Reduction

Mit Hilfe von Load Reduction wird der gesamte Lastverlauf am Tag gesenkt. Durch die Steigerung der Energieeffizienz durch Verwendung von neuen Technologien (z.B. Einsatz von Energiesparlampen) wird gleichzeitig der Quality of Service gehalten. Bei der Energieeinsparung dagegen wird auf bestimmte Verbraucher einfach verzichtet (weniger Lampen einschalten), was zu einem Verlust beim Quality of Service führt.

4.5.4 Erweiterung klassischem Lastmanagements

Durch die klassischen Lastmanagementverfahren ist aber die Einspeisung EE nicht bedacht. So können in einem Gebäude mittlerweile Lastspitzen beispielsweise gewollt sein, um die maximal installierte Leistung an EE auszunutzen. Bei Erzeugung von positiver Regelenergie zum Beispiel durch Speicher kann zusätzlich Last aufgenommen werden, was das sonst nötige Ausschalten von z.B. Photovoltaikanlagen auf dem Gebäudedach verhindern könnte. Für einen möglichst guten Fahrplan im Gebäude sind außerdem Wetterdaten für das BEMS von Bedeutung sofern davon abhängige Erzeuger installiert sind, was die Planung des Lastmanagements mit beeinflussen sollte.
Laut der Agora Studie „Lastmanagement als Beitrag zur Deckung des Spitzenlastbedarfs in Süddeutschland“ besteht ein Regelenergiepotenzial durch Lastmanagement bei den industriellen Querschnittstechnologien und energieintensiven Prozessen von über einem GW in Süddeutschland, das über einen Zeitraum von 30 Minuten bis zu zwei Stunden aktiviert werden kann [4].

4.6 BEMS im Bezug auf die PG

Auch in der Projektgruppe muss ein möglichst gutes Management entwickelt werden, damit der Energieverbrauch möglichst optimal gehandelt wird. Die eigentliche Hardware wird dabei zwar simuliert sein, aber eine möglichst getreue Simulation soll dasselbe Szenario wie in der Realität widerspiegeln. Die Entwicklung des Gebäudemanagements unter Verwendung von Heuristiken oder anderen Methoden ist dabei ein zentraler Bestandteil der Aufgabe. Zwar wird von dem zu entwickelnden System keine Regelenergie nach außen abgegeben werden, dennoch in die momentane Problematik des energieeffizienten Verbrauchs eingebettet sein. Eine möglichst gute Simulation und optimierter Verbrauch innerhalb eines Gebäudes ist das vorgegebene Ziel der Projektgruppe. Hierfür können bereits entwickelte Gebäude-Energiemanagementsysteme wie beispielsweise das CAE von Cylon als Orientierungshilfe dienen und ein Beispiel darstellen. Wie auch beim System von Cylon, werden die (steuerbaren) Stromverbraucher durch eine Zentraleinheit kontrolliert und ihr Verbrauch reguliert. Der Administrator soll diese Regularien einstellen und später ein genaues Ergebnis über Erzeugung und Verbrauch des Gebäudes haben und nötigenfalls mit entsprechenden Maßnahmen darauf reagieren können.

4.7 Zusammenfassung

Durch den Einsatz von BEMS kann nicht nur eine Kostenreduktion für den Nutzer erzeugt werden. Neben der Stabilisierung des Stromnetzes wirkt sich eine effizientere Energienutzung auch positiv auf die Umweltverschmutzung aus. Steigender Rege-

lenergiebedarf in Deutschland durch den wachsenden Anteil an EE könnte durch den allgemeinen Einsatz von Lastmanagement teilweise gedeckt werden. Die bisher aufgeführten Lastmanagementverfahren werden zwar bereits verwendet müssen aber für ein BEMS das gleichzeitig Erzeuger aus dem Bereich EE des Gebäudes kontrollieren soll optimiert werden. Durch fluktuierende Einspeisung EE ist zusätzliche Regelenergie nötig um sie optimal zu nutzen.

Literaturverzeichnis

1. Energiewende - Anteil erneuerbarer Energien wächst weiter. Website des Presse- und Informationsamts der Bundesregierung. URL www.bundesregierung.de/Content/DE/Artikel/2014/01/2014-01-13-bdew-energiebilanz-2013. Letzter Zugriff 12.06.2014
2. Bundesregierung, D.: Regelenergie -leistung. Webseite. URL http://www.mark-e.de/Home/Geschaeftskunden/Beratung-342-1/FAQ-539-1/Regelenergie-leistung.aspx. Letzter Zugriff 12.06.2014
3. Dutch, R.: Shell Global Scenarios to 2025. Royal Dutch/Shell Group (2005)
4. Energiewende, A.: Dezentrales Lastmanagement zum Ausgleich kurzfristiger Abweichungen im Stromnetz. Agora Energiewende (2013)
5. GmbH, C.: Cylon. Webseite. URL www.cylon.com. Letzter Zugriff 12.06.2014
6. Kamper, A.: Dezentrales Lastmanagement zum Ausgleich kurzfristiger Abweichungen im Stromnetz. KIT Scientific Publishing (2010)
7. Kraftwerke, N.: Regelenergie. Webseite. URL http://www.next-kraftwerke.de/wissen/regelenergie. Letzter Zugriff 12.06.2014
8. Michael Sonnenschein Barbara Rapp, J.B.: Demand-Side Management und Demand Response. EW Medien und Kongresse GmbH (2010)
9. NRW, E.: Energieeffizienz in Krankenhäusern. Webseite. URL http://www.energieagentur.nrw.de/unternehmen/energieeffizienz-in-krankenhaeusern-4058.asp. Letzter Zugriff 12.06.2014
10. on behalf of Sustainable Energy Ireland (SEI), N.C.G.: Building Energy Manager's Resource Guide. Sustainable Energy Ireland (2013)

Kapitel 5
Model Predictive Control

Eugen Langolf

Zusammenfassung In diesem Kapitel wird das Konzept der *Model Predictive Control* (MPC) vorgestellt und seine mögliche Anwendung für die PG *Hardwarebasierte Simulation energieautonomer* Gebäude betrachtet.

5.1 Einleitung

MPC ist eine Art der Regelung. Das bedeutet es werden Eingangswerte eines Prozesses kontrolliert um bestimmte Ausgangswerte zu erzielen. MPC verwendet ein Modell um damit den erwarteten weiteren Verlauf des zu regelnden Prozesses vorherzusagen. Diese Vorhersage wird genutzt um gezielt Eigenschaften des Prozessverlaufs zu optimieren und entsprechende Eingangswerte zu wählen. Dabei können auch Einschränkungen der Eingangs- und Ausgangswerte berücksichtigt werden. Die Hauptanwendung findet MPC in geregelten Prozessen, bei denen mehrere Eingangs- und Ausgangswerte interagieren und durch Werteeinschränkungen begrenzt sind, wie Beispielsweise chemische Prozesse.

5.2 Überblick

Nach Qin und Badgwell können die Ziele eines MPC wie folgt zusammengefasst werden:

- Werteeinschränkungen (*Constraints*) für Eingangs- und Ausgangswerte einhalten.

Carl von Ossietzky Universität Oldenburg
E-mail: eugen.langolf@uni-oldenburg.de

- Optimiere einen Teil der Ausgangswerte und halte die restlichen Werte in bestimmten Bereichen.
- Halte die Veränderung der Eingabewerte gering.
- Kontrolliere so viele Werte wie möglich, falls ein Sensor ausfällt.

Drei wesentliche Eigenschaften heben MPC von anderen Reglern ab. Zum einen die Vorhersage des Prozessverlaufs und die daraus resultierende Optimierung. Des weiteren die Fähigkeit dieses bei Systemen mit mehreren Ein- und Ausgaben (MIMO System, multiple input multiple output) mit Interaktionen zu schaffen. Und schließlich die Einhaltung von *Constraints*.

5.3 Aufbau

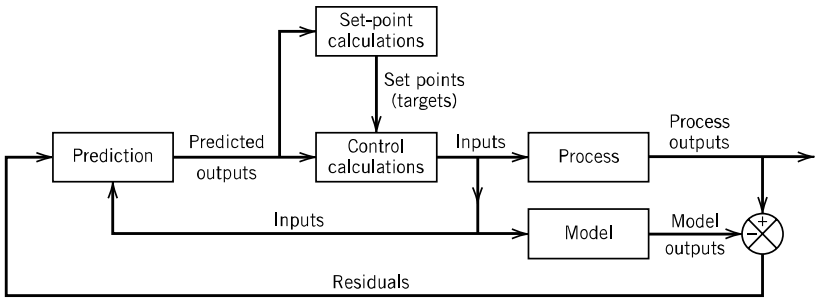


Abb. 5.1 MPC Aufbau (Seborg)

Der Aufbau eines typischen MPC kann dem Blockdiagramm in Abbildung 5.1 aus [1] entnommen werden. Zwei Berechnungen finden regelmäßig statt. Zunächst werden die aktuell angestrebten Sollwerte berechnet. Dies geschieht in der *Set-point calculation*. Dann wird der nächste Kontrollschritt ermittelt um die berechneten Sollwerte zu erreichen (*Control calculations*). Diese Berechnungen werden in jedem Schritt mehrmals ausgeführt und mithilfe einer Vorhersage (*Prediction*) wird eine Reihe von Kontrollschritten ermittelt. Die Vorhersage geschieht dabei mit Hilfe eines internen Prozessmodels und soll den wahrscheinlichen weiteren Verlauf des Systems darstellen. Beide Berechnung optimieren jeweils eine Funktion bei der je nach Anwendung verschiedene Faktoren gewichtet werden. Der erste Kontrollschritt wird an den Prozess (*Process*) und ein *Model* übergeben. Alle anderen Kontrollschritte werden verworfen, diese dienten nur der Vorhersage. Die Differenz zwischen den tatsächlichen Ausgangswerten des Prozesses und den vorhergesagten Werten des Modells wird als *Residuals* an die Vorhersage zurück geführt um Abweichungen zu verringern.

5.4 Ablauf

Der Ablauf einer Kontrollberechnung wird in Abbildung 5.2 dargestellt.

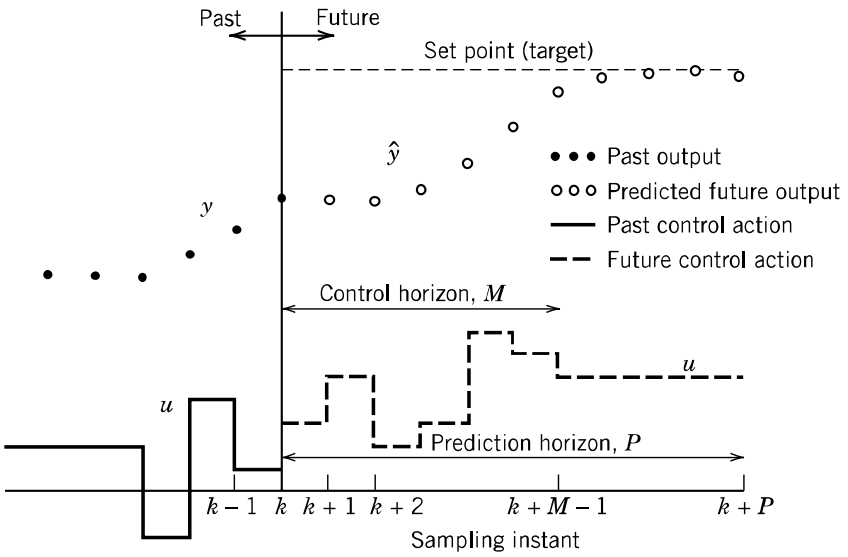


Abb. 5.2 MPC Ablauf (Seborg)

Im k-ten Regelzeitpunkt werden M Kontrollschritte berechnet. Mit Regelzeitpunkt ist der Zeitpunkt gemeint, in dem der Regler Gelegenheit bekommt die Eingangswerte zu verändern. Ein Kontrollschritt dauert solange, wie die Zeit zwischen zwei aufeinander folgenden Regelzeitpunkte beträgt. Diese Menge an Kontrollschritten wird auch Kontrollhorizont genannt (*Control horizon, M*) und beschreibt wie viele verschiedene Schritte berechnet werden bevor ein konstanter Wert für die Eingabe angenommen wird. Anschließend wird bei konstantem Eingabewert betrachtet wie sich der weitere Verlauf der Ausgangswerte entwickelt. Der Zeitraum vom aktuellen Zeitpunkt bis zum Ende dieser Betrachtung wird Vorhersage Horizont (*Prediction horizon, P*) genannt. Die Menge y beschreibt die Menge der vergangen Ausgangswerte. Im Falle von mehreren Ausgaben ist y eine Menge von Vektoren. Die Menge u ist die Menge sowohl vergangener als auch zukünftiger Eingangswerte und der Wert $u(i)$ bezeichnet den Eingabewert zum Regelzeitpunkt i. Auch hier handelt es sich im Falle von mehreren Eingaben um Vektoren aller Eingaben. Ziel der Kontrolle ist es, jeden Ausgangswert auf einen angestrebten Wert zu bringen (*Set point (targert)*).

5.5 Internes Modell

Die Vorhersage, welche zwingend für den MPC erforderlich ist, wird mittels eines internen Modells berechnet. Dieses Modell muss den geregelten Prozess möglichst genau darstellen, da bessere Vorhersagen eine bessere Regelung ermöglichen. Andererseits wird dieses Modell häufig verwendet um die nächsten Zustände des Systems zu bestimmen. Hier gilt es also einen Kompromiss zwischen Genauigkeit und Berechenbarkeit zu wählen. Historisch wurden dabei meist lineare Modelle verwendet, da diese zwar nicht exakt das meist nicht lineare System darstellen, aber sehr nah am korrekten Wert sind und eine einfache Berechnung ermöglichen. Mit der aktuellen Rechenleistung gängiger PCs ist es je nach System nicht zwingend notwendig ein lineares Modell zu verwenden.

5.5.1 Beispiel eines linearen Modells

Eine relativ simple Methode die nächsten Ausgangswerte des Systems vorherzusagen ist es die aktuellen Ausgangswerte und die gewählten Eingangswerte jeweils mit einer Matrix zu multiplizieren und zu addieren. Um also die Ausgangswerte y zum Zeitpunkt $k + i + 1$ zu berechnen, würde man die Ausgangswerte zum vorherigen Zeitpunkt $y(k + i)$ mit einer quadratischen Matrix A multiplizieren und die Eingangswerte zu diesem Zeitpunkt $u(k + i)$ mit der Matrix B multipliziert dazu addieren. Die Formel wäre also:
$y(k + i + 1) = A * y(k + i) + B * u(k + i)$
Bei dieser Darstellung des Systems bestimmen die Matrizen A und B das gesamte System.

5.5.2 Ermittlung des Modells

Um ein korrektes Modell des Systems zu ermitteln wären zwei Wege denkbar. Einerseits könnte anhand des Systemaufbaus und der Theorie der einzelnen Komponenten ein theoretisches Gesamtmodell erzeugt werden. Eine andere Option wäre die experimentelle Ermittlung der Modellteile. Diese zweite Methode lässt sich unter Umständen nicht anwenden, falls Testläufe mit unvorhergesehenen Werten zu Beschädigungen einer zu teuren Anlage führen könnten.

5.6 Set Point Berechnung

Mit *Set Point* wird der angestrebte Wert für Ein- und Ausgabe bezeichnet. In der Regel gibt es durch eine höhere Instanz eine Vorgabe für die Ein- und Ausgabewerte

uStrich und *yStrich*. Allerdings können diese Werte möglicherweise nicht erreicht werden, da der aktuelle Zustand des Prozesses dies nicht zulässt. Deshalb werden in der *Set Point Calculation* diese Werte so berechnet, dass sie von der *Control Calculation* erreicht werden können und eine Kostenfunktion optimiert wird.

5.6.1 Kostenfunktion für Set Points

Die Kostenfunktion besteht aus mehreren gewichteten Komponenten. Es handelt sich meist um eine Minimum Funktion, abhängig von den gewählten Set Points für die Ein- und Ausgangswerte. Die Set Points werden so gewählt, dass diese Funktion ein minimales Ergebnis ergibt. Eine mögliche Funktion wäre

$$\min_{u_{sp}, y_{sp}} J_s = c^T u_{sp} + d^T y_{sp} + e_y^T Q_{sp} e_y + e_u^T R_{sp} e_u + S^T T_{sp} S$$

Die Faktoren $c, d, Q_{sp}, R_{sp}, T_{sp}$ gewichten verschiedene Aspekte der Set Points. Diese variieren je nach Anwendung und werden bei der Konzipierung des MPC ermittelt. u_{sp} ist der Vektor aus den gewählten Set Points für die Eingangsgrößen. Der Vektor y_{sp} bezeichnet die gewählten Set Points für die Ausgangsgrößen. Diese beiden Werte dienen der späteren Berechnung der tatsächlich eingestellten Eingangswerte und ergeben sich aus der Minimierung der Kostenfunktion. y_{ref} sind für den MPC vorgegebene Ziele für die Ausgangswerte. Diese kommen von einer deutlich seltener ausgeführten Berechnung, wie zum Beispiel einem Tagesplan oder einem vorher geplanten Herstellungsverfahren. Die Abweichung von der Vorgabe, also die Differenz zwischen dem vorgegebenen und dem gewählten Wert wird mit $e_y = y_{sp} - y_{ref}$ bezeichnet. Entsprechend gibt es die Werte x_{ref} und e_x für die Eingangswerte. S enthält die *step-response* Koeffizienten der nächsten M Schritte.

Beispiele

Maximaler Profit Wenn die Zielfunktion maximalen Profit anstreben soll, müssen die Gewichtungsfaktoren so gewählt werden, dass der Zusammenhang $J_s = -Profit$ gilt. Die Abweichung vom vorgegebenen Wert spielt also keine Rolle, daher können $Q_{sp} = 0, R_{sp} = 0$ gesetzt werden. Die restlichen Faktoren müssen so gewählt werden, dass der beschriebene Zusammenhang gilt.

Minimale Abweichung Ist hingegen die wesentliche Aufgabe des MPC das System möglichst nah an der Vorgabe zu halten, so wären die konkret gewählten Werte unwichtig. Daher würde man die Faktoren $c = 0, d = 0$ setzen. Je nachdem welche Werte wichtiger sind oder welchen Einfluss die step-responce spielt, müssen die anderen Faktoren gesetzt werden.

Maximale Produktion Soll nun die Produktion das einzige Ziel der Optimierung sein und stellt u_1 die Produktionsrate pro Schritt dar, so wird das Verhältnis $J_s =$

$-u_1$ angestrebt. Dazu wird $c_1 = -1$ gesetzt und alle anderen Gewichtungen sind 0.

5.7 Input Berechnung

Aus den bestimmten *set points* werden nun die m nächsten Kontrollaktionen, d.h. die nächsten m Eingangswerte berechnet. Dies erfolgt durch Optimierung einer Zielfunktion. Die Kontrollaktionen werden so gewählt, dass die besagte Zielfunktion den minimalen Wert ergibt.

5.7.1 Zielfunktion

Drei Faktoren spielen eine wesentliche Rolle für die Güte der Kontrollaktionen.

Ausgangsabweichung $\hat{E}(k+1)$ bezeichnet die Abweichung zwischen den set points der Ausgangswerte und den vorausgesehenen Werten für die Ausgangswerte über den Vorhersage Horizont. Diese gewichtet zu Minimieren ist ein mögliches Ziel eines MPC.

Änderung der Eingangswerte Ein anderes Ziel könnte es sein die Eingangswerte möglichst wenig zu Verändern. Das heißt die Differenz $\Delta u(k) = u(k) - u(k-1)$ für die berechneten m Kontrollaktionen soll gewichtet minimiert werden.

Eingangsabweichung Ein weiteres Ziel der Zielfunktion kann es sein die Abweichung zu den in der Kostenfunktion ermittelten Set Points für die Eingangswerte zu verringern. Das bedeutet der Betrag der Differenz $u_{sp}(k+i) - u(k+i)$ soll möglichst gering gehalten werden für den gesamten Kontrollhorizont.

Einer oder mehrere dieser Faktoren können zu einer Zielfunktion zusammengefasst werden. Zum Beispiel:
$\min_{\Delta U(k)} J = \hat{E}(k+1)^T Q \hat{E}(k+1) + \Delta U(k)^t R \Delta U(k)$
Diese Zielfunktion vereint die ersten beiden Faktoren und gewichtet sie mit den Matrizen Q und R.

5.8 Werteeinschränkungen

Es gibt zwei Arten von Werteeinschränkungen (*constraints*). Dabei kann es sich sowohl um untere als auch obere Schranken für Eingangs- und Ausgangswerte handeln.

5.8.1 hard constraints

Zum einen gibt es harte Einschränkungen (*hard contraints*). Diese dürfen auf keinen Fall überschritten werden dürfen. Um diese Einzuhalten müssen Set Points und Kontrollaktionen so gewählt werden, dass sowohl die vorhergesagten Eingangs- als auch Ausgangswerte all ihre *hard constraints* erfüllen. Auswahlen, die gegen *hard constraints* verstoßen stellen keine gültige Lösung dar.

5.8.2 soft constraints

Schwache Einschränkungen (*soft constraints*) stellen ungünstige Zustände des Systems dar, die zwar kein direktes Ausschlusskriterium für eine Lösung sind, aber sich negativ auf das System auswirken und nicht lange beibehalten werden sollen. Um dies zu Erreichen kann die Abweichung aus den *soft constraints* sich mit einem Faktor entweder in der Kostenfunktion als auch in der Zielfunktion negativ auf das Ergebnis auswirken. So werden Lösungen in denen gegen *soft constraints* verstoßen wird gemieden. In beiden Fällen sollten die Verstöße entsprechend ihrer Auswirkung gewichtet werden.

5.9 Relevanz für die PG

In [2] wird eine mögliche Verwendung von MPC in Energiesystemen betrachtet. Dort wird ein System aus verschiedenen erneuerbaren Energieerzeugern und variablen Verbrauchern beschrieben. Außerdem dienen Energiespeicher zum Ausgleich von Differenzen zwischen Erzeugern und Verbrauchern. Der komplette Tagesverlauf wird mittels Verbrauchs- und Wettervorhersagen im voraus geplant. Sowohl die Vorhersage der Verbraucher kann je nach Anwendung von den realen Werten abweichen, da Teile des Verbrauchs einfach variabel sind. Die Wettervorhersage dient zur Ermittlung von wahrscheinlich erzeugen Energiewerten, allerdings sind auch Wettervorhersagen nie komplett korrekt. Der initiale Tagesplan dient dazu den Bedarf an externer Energie und die damit verbundenen Kosten zu reduzieren. Dazu werden die Energiespeicher zu geeigneter Zeit gefüllt und verwendet. Spontane Abweichungen können mit einem Tagesplan nicht kompensiert werden. Deshalb wird ein MPC verwendet um unvorhergesehene Abweichungen sowohl bei Verbrauchern als auch bei Erzeugern zu kompensieren und den Bedarf an externer Energie möglichst gering zu halten.

Da unsere PG ein ähnliches Ziel verfolgt, könnten auch wir einen Tagesplan in Verbindung mit einem MPC verwenden um den Energieverbrauch oder die Zufuhr von externer Energie zu verringern.

5.10 Zusammenfassung

Es wurden MPCs als eine Form von Reglern vorgestellt. Seine Komponenten und der Verlauf wurden erläutert. MPCs sind gut geeignet für Regelaufgaben mit mehreren Ein- und Ausgangswerten und könnten in unserem Projekt aufgrund ihrer Optimierungsmöglichkeiten gut verwendet werden.

Literaturverzeichnis

1. et al., S.: Process Dynamics and Control (2011)
2. Michele Arnold, G.A.: Model predictive control of energy storage including uncertain forecasts. Tech. rep., ETH Zurich

Kapitel 6
PC-Based Automation/eXtended Automation

Connor Fibich

Zusammenfassung Dieser Beitrag gibt eine Einführung in die PC-Based Automation. Dabei wird zunächst der Aufbau von Automationsanlagen vorgestellt und zwischen zwei Umsetzungsmöglichkeiten verglichen. Anschliessend wird eine Umsetzungsmöglichkeit von PC-Based Automation mit TwinCAT vorgestellt. Weiterhin wird auf Realtime-Ethernet insbesondere EtherCAT eingangen.

6.1 Einleitung

Im Rahmen der Projektgruppe soll eine hardwarebasierte Simulation eines energieautonomen Gebäudes erstellt werden. Diese Simulation wird mittels einer PC-basierten Automationsanlage realisiert. Diese Ausarbeitung beschreibt solche Automationsanlagen und zeigt in einen Vergleich die UNterschiede zwischen PLC und PC basierten Systemen. Darüberhinaus soll diese Arbeit eine Einführung in die "eXtended Automation" und EtherCAT geben. Diese beiden Technologien sind grundlegend für die verwendung des PC-basierten Automationssystem, welches der Projektgruppe zur Verfügung steht.

6.2 Automationsanlagen

Abläufe müssen in der Industrie effizient und sicher durchgeführt werden. Die Ausführung dieser Abläufe durch Menschen ist aufgrund der Effizienz, die stark vom Personal abhängt, sowie wegen der Sicherheit des Personals (z.B. in für Menschen nicht geeigneten Umgebungen) nicht gewünscht. Deshalb werden Automationsanlagen verwendet. Diese sind in der Lage, solche Abläufe effizient und sicher auszuführen.

Carl von Ossietzky Universität Oldenburg
E-mail: connor.fibich@uni-oldenburg.de

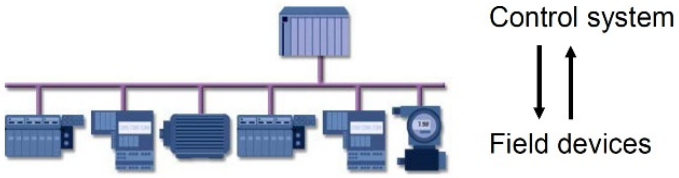


Abb. 6.1 Aufbau einer Automationsanlage (Quelle: `http://us.profinet.com/technology/profibus/`)

Abbildung 6.1 zeigt den Aufbau von Automationsanlagen. Sie bestehen im Wesentlichen aus drei Bestandteilen: dem Kontrollsystem, den Feldgeräten und dem Feldbus (in der Abbildung über die Verbindungen zwischen Kontrollsystem und den Feldgeräten dargestellt).

Das Herzstück jeder Automationsanlage ist das Kontrollsystem, das die Feldgeräte steuert und überwacht, mit denen die Aufgabe des Systems erfüllt wird. Dieses System wird mit Hilfe von PLCs (programmable logic controller) oder PCs (Personal Computer) realisiert. In der Vergangenheit waren PLCs die gängige Basis für Kontrollsysteme. Aufgrund von Weiterentwicklungen beider Systeme, die Aspekte des einen Systems auf das jeweils andere überführten, hat man heute die Wahl zwischen beiden Systemen.

Die Feldgeräte sind die Komponenten der Automationsanlage, mit denen der eigentliche Automationsprozess/die Aufgabe der Anlage realisiert und überwacht wird. Sie erhalten Steuersignale von dem Kontrollsystem und geben Informationen über ihren Zustand an das Kontrollsystem zurück.

Für die Übertragung dieser Steuersignale bzw. Zustandsinformationen wird ein Feldbus benötigt. Dieser realisiert die Verbindung zwischen den Feldgeräten und dem Kontrollsystem.

6.2.1 Grundlagen: PLC

PLCs werden seit 1970 für die Kontrolle von Automationssystemen verwendet. Sie wurden als eine effizientere, flexiblere und zuverlässigere Alternative zu Schaltkästen und Relais-Panels entwickelt, deren Aufgabe sie übernehmen. Dementsprechend wurde auch die Programmiersprache und Struktur in Anlehnung an die Schaltkästen und Relais-Panels entworfen, die durch den PLC ersetzt werden sollten. PLCs bieten eine sehr robuste und konsistente Leistung, auch in anspruchsvollen Umgebungen mit äußeren Einflüssen (z.B. elektromagnetische Störausstrahlung oder Vibration). Im Laufe der Zeit übernahmen PLCs einige Funktionen des PCs, wie zum Beispiel Web-Server. [4]

6.2.2 Grundlagen: PC

Der PC wurde hauptsächlich für komplexe Berechnungen, Überwachung des Systems und als User-Interface für PLC verwendet. Wegen ihrer empflindlichen Bauteile, wie Festplatten und Lüfter, konnten PCs nicht denselben Umgebungen wie PLCs standhalten. In den 90er Jahren stieg jedoch die Anzahl der Automationssysteme die PCs verwendeten. Weiterentwicklungen des PCs führten dazu, dass dieser die Funktionalität des PLCs übernehmen konnte. Dabei wurden Möglichkeiten gefunden den PC den Einsatzumgebungen des PLCs anzupassen. [4]

6.3 Vergleich PC / PLC

Bei der Auswahlentscheidung zwischen PLC und PC kann der Entwickler nach folgenden Kriterien die Vor- und Nachteile der beiden Umsetzungsvarianten abwägen und dabei individuell auf das System abgestellte Prioritäten zugrundelegen:

- Operation
- Robustness
- Serviceability
- Programming

[4]

Operation

Unter dem Kritierum Operation wird untersucht, wie zuverlässig beide Systeme betrieben werden können.

Der PLC wurde speziell für Automationssysteme entwickelt. Deshalb besitzen PLCs ein Realzeit-Betriebssystem mit einem speziell dem Echtzeit-Betrieb gewidmeten Prozessor. Dadurch garantieren PLCs ein hohes Maß an Zuverlässigkeit für Kontrollsysteme.

Bei der Entwicklung des PCs hingegen wurde nicht das Ziel verfolgt, Realzeitaufgaben zu erledigen. Mit Hilfe eines Realzeit-Kernels oder Realzeit-Betriebssystems jedoch erreicht der PC dasselbe Maß an Zuverlässigkeit wie PLCs.

Robustness

Robustness beschreibt, wie gut ein System den unterschiedlichen Umwelteinflüssen der Einsatzumgebung standhalten kann.

Da PLCs entwickelt wurden, um als Kontrollsystem für Automationsanlangen in einem industriellen Umfeld eingesetzt zu werden, wurden diese speziell entwickelt, um den Besonderheiten dieser Umgebungen standhalten zu können.

Der PC hingegen wurde für den Betrieb in kontrollierten Umgebungen (z.B. im Büro) entwickelt. Dieses Umfeld erlaubt die Verwendung von Lüftern und Festplatten, welche wiederum eine höhere Leistungsaufnahme des Prozessors erlauben bzw. mehr Speicherkapazität für Programme bieten. Im industriellen Umgebungen sind solche beweglichen Teile zu störungsanfällig und deshalb oftmals ungeeignet. Lüfterlose Kühlungsysteme und neuere Entwicklungen wie Solid-State-Drives ersetzen solche beweglichen Teile aus dem PC und erlauben damit den Betrieb von PCs in Umgebungen in denen bisher nur PLCs eingesetzt werden konnten. Eine weitere Option, die den Einsatz von PCs auch unter Verwendung empfindlicherer Bauteile ermöglicht, ist die Anpassung des Umfelds durch Einbau des PCs in Schränke, die eine kontrolliertere Umgebung zur Verfügung stellen.

Serviceability

Ein weiterer wichtiger Faktor, der berücksichtigt werden muss, ist die Wartbarkeit des Kontrollsystems.

PLCs bieten durch ihren modularen Aufbau eine gute Struktur, um Wartungsarbeiten in einfacher Weise vornehmen zu können. Oft sind auch lange nach Anschaffung noch baugleiche Ersatzteile verfügbar, so dass Wartungsarbeiten an der Hardware keine Anpassung der softwareseitigen Implementation des PLCs erfordern.

Für den PC sind baugleiche Ersatzteile oft nur über eine kurze Zeitdauer verfügbar. Aufgrund der Verfügbarkeit müssen deshalb eventuell neuere/alternative Bauteile verwendet werden, die wiederum die Notwendigkeit einer Anpassung der Automationssoftware nach sich zieht. Der PC bietet jedoch zusätzlich die Möglichkeit, bestimmte Geräte zur Laufzeit auszutauschen (hot swap).

Programming

Neben den Anforderungskriteren, die sich auf technische Faktoren beziehen, ist die Art und Weise, wie PLCs und PCs programmiert werden, ein weiteres wichtiges Kriterium.

Die Unterschiede zwischen PLC und PC zeigen sich bereits beim zugrundeliegenden Konzept für die Ausführung der Programme. PLCs nutzen eine Mixtur aus scan-based und event-driven Programmierung, während PCs meist nur event-driven programmiert werden. Scan-based Programme können länger für die Ausführung benötigen, da Aufgaben mit höherer Priorität im Zyklus zuerst behandelt werden.

Auch bei den unterstützten Programmiersprachen bestehen große Unterschiede. Programme für PLCs werden mit Programmiersprachen erstellt, die in IEC 61131-3

definiert sind. In IEC 61131-3 werden fünf Programmiersprachen für PLCs definiert. Davon werden zwei Sprachen textbasiert und drei graphisch programmiert. Diese Programmiersprachen sind angelehnt an die Art der Bauteile, die von den PLCs ersetzt wurden. Unter anderem werden in der Sprache LD (Ladder Diagram) Programme graphisch aufgebaut, die Elektro-Schaltplänen ähneln. Teilweise werden auch herstellerspezifische Sprachen verwendet.

Für die Programmierung von PCs steht ein breites Spektrum an Programmiersprachen zur Verfügung. Zu diesen Programmiersprachen gehören C, C++, .NET und auch andere Sprachen, die für Realzeit-Programmierung unter Umständen jedoch nicht geeignet sind.

Fazit des Vergleichs

Der Vergleich zeigt deutlich, dass PLC und PC für die Kontrolle einer Automationsanlage auf einem ähnlichen Niveau geeignet sind. Neben den unterstützen Programmiersprachen sind deshalb die Kosten, die für die Anschaffung und Wartung solcher Systeme entstehen, ein wichtiger Entscheidungsfaktor.

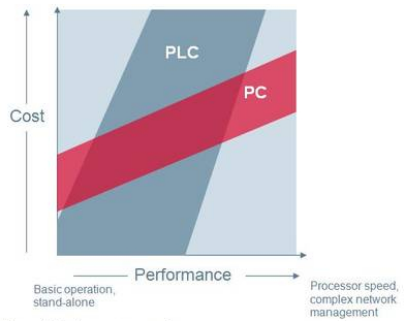


Abb. 6.2 Leistung vs. Kosten[4]

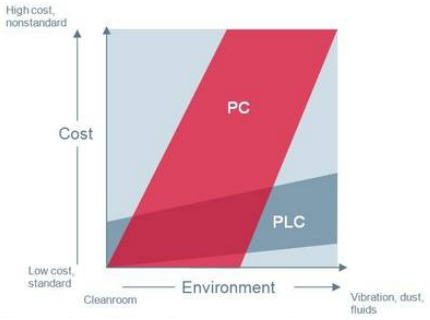


Abb. 6.3 Umgebung vs. Kosten[4]

Abhängig vom Aspekt, unter denen PLC und PC miteinander verglichen werden, entstehen unterschiedliche Eindrücke über die Kosten die durch die jeweiligen Systeme entstehen. Abbildung 6.2 stellt die Kosten von PLC und PC in Bezug zu der benötigten Leistung. Hier wird deutlich, dass PCs bei Anwendungen, die nur wenig Leistung benötigen, höhere Kosten als ein PLC verursachen. Wird mehr Leistung benötigt, steigen die Kosten für die Umsetzung mit PLC deutlich über die Umsetzungskosten mit PC. Bei der Gegebenüberstellung der Kosten in Hinsicht auf die Einsatzumgebung entsteht ein gegenteiliger Eindruck (Abbildung 6.3). Hier verursachen PLC und PC für unanspruchsvolle Umgebungen jeweils ähnlich geringe Kosten. Je anspruchsvoller die Einsatzumgebung ist, desto mehr steigen die Kosten

eines PCs, der in dem Umfeld betrieben werden kann. Die Kosten für PLC steigen für den Einsatz in solchen Umgebungen nur geringfügig an.[4]

6.4 eXtended Automation/TwinCAT

Für die hardwarebasierte Simulation steht der Projektgruppe ein PC-basiertes Automationssystem der Firma Beckhoff zur Verfügung.

Automationsprogramme für dieses System können mit Hilfe der Automatisierungssuite "TwinCAT 3 - eXtended Automation" entwickelt werden.

Mit dieser Automatisierungssuite können PC-basierte Systeme in Echtzeitsteuerungen umgewandelt werden. Auf dieser Echtzeitsteuerung können mehrere PLC-, NC (Numerical Conrol), CNC- (Computerized Numerical Conrol) und Robotik-Laufzeitsysteme ausgeführt werden. Weiterhin bietet die Automatisierungssuite eine Umgebung mit der Automationssysteme erstellt werden können.

Diese beiden Bestandteile von TwinCAT werden als eXtended Automation Engineering und eXtended Automation Runtime bezeichnet. [2]

6.4.1 eXtended Automation Engineering

eXtended Automation Engineering bezeichnet die Umgebung in der TwinCAT-Module entwickelt werden. Durch die Integration in Visual Studio kann dessen Funktionalität, wie zum Beispiel die Anbindung an Quellcodeverwaltungssoftware genutzt werden. Neben den Funktionen von Visual Studio erlaubt TwinCAT die Steuerungsprogrammierung mit den Programmiersprachen definiert in IEC 61131-3. Über die für PLCs entwickelten Sprachen hinaus ist es auch möglich, Automations-Module in C/C++ zu erstellen. Solche in C oder C++ entwickelte Echtzeitprogramme können wie gewohnt durch die von Visual Studio bekannten Wege debuggt werden.

Weiterhin besteht eine Anbindung an Matlab/Simulink. Diese erlaubt die Verwendung eines bereits existerenden Standardwerkzeugs für wissenschaftliche und messtechnische Anwendungen zur Erstellung von TwinCAT-Modulen. Mit Hilfe einer Debug-Schnittstelle können Module, die in Simulink erstellt wurden, zur Laufzeit in der Echtzeitumgebung angepasst werden. Module, die auf einer Simulink-Simulation basieren, können dabei unabhängig von Simulink in TwinCAT genutzt werden.

Abbildung 6.4 zeigt den Aufbau des Engineering Environment von TwinCAT 3. Dabei wird dagestellt welche Teile des Environments in Visual Studio integriert wurden und wie diese Untereinander und mit der Automation Runtime verbunden sind.

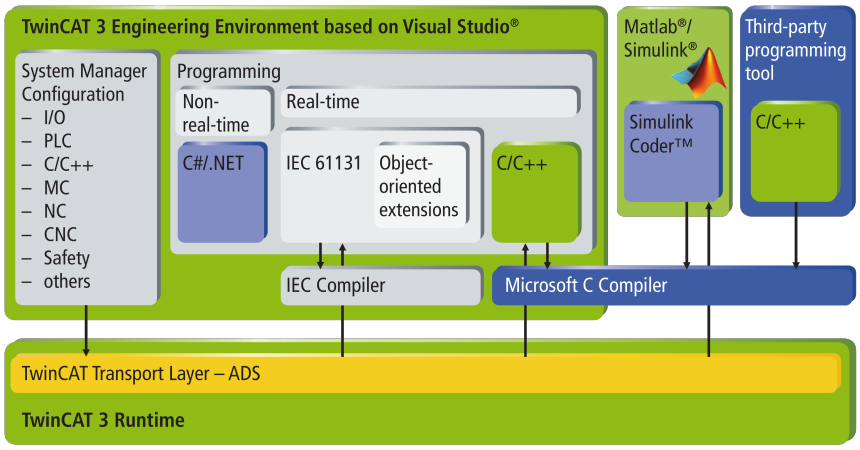


Abb. 6.4 Engineering Environment von TwinCAT[2]

6.4.2 eXtended Automation Runtime

Die eXtended Automation Runtime ist die von TwinCAT bereitgestellte Umgebung, in der TwinCAT-Module ausgeführt werden können. Die Runtime verfügt dabei über einen Realzeit-Kernel, auf dem beliebige TwinCAT-Module zuverlässig ausgeführt werden können. Über den Automation Device Driver können die Module auf alle wichtigen Feldbusse zugreifen.

Abbildung 6.5 zeigt den Realzeit-Kernel auf dem TwinCAT-Module ausgeführt werden. Diese Module können wiederum auf Funktionen anderer Module zurückgreifen. Über den TwinCAT Transport Layer erhält das Engineering Environment

6.4.3 Verwendung von TwinCAT [1]

In diesem Abschnitt soll beispielhaft veranschaulicht werden, wie C++-Module in TwinCAT definiert und debuggt werden. **Erstellen neuer C++-Module:**
In TwinCAT-Projekten werden Programmierobjekte des Projekts nach ihrem Typ in einer Baumstruktur organisiert (zu sehen in Abbildung 6.6). Um ein neues C++-Modul zu erstellen, wird in der Struktur "C++" ausgewählt und über dessen Kontextmenü durch Auswahl von "Add New Item..." ein Fenster zum Erstellen neuer C++-Objekte geöffnet. In diesem Fenster wird nun das Template "TwinCAT Driver Project" ausgewählt. TwinCAT bietet daran anschließend einige Templates für verschiedene C++-Module an, die als Basis für das zu erstellende Modul genutzt werden können. Im vorliegenden Beispiel wird das Template "TwinCAT Module Class with Cyclic IO" verwendet.

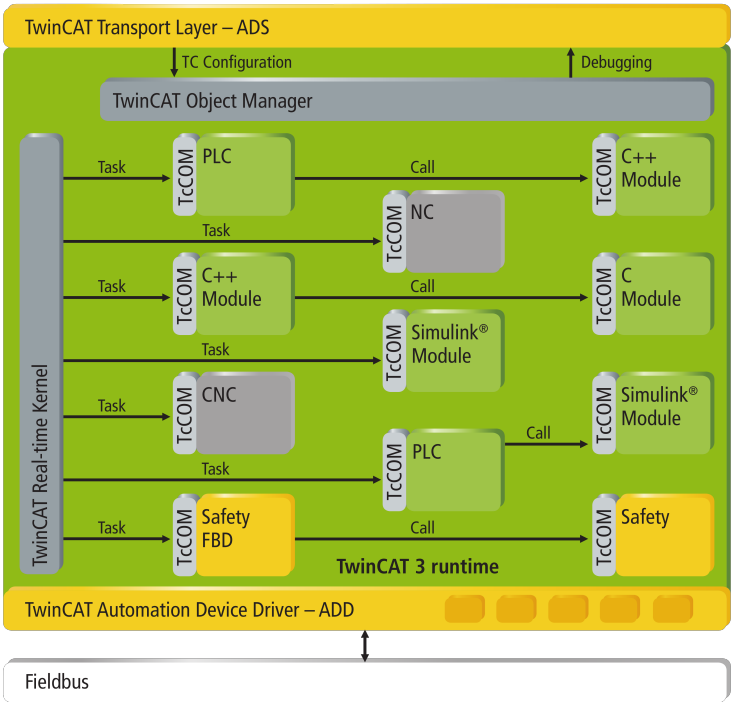


Abb. 6.5 Runtime von TwinCAT[2]

TwinCAT erzeugt nun die Strukturen für das C++-Modul. In der .cpp-Datei kann nun die C++-Implementation des Moduls erstellt werden. Dabei muss Programmcode, welcher zyklisch ausgeführt werden muss, der bereits angelegten Methode CycleUpdate() hinzugefügt werden.

Nachdem ein Modul erstellt wurde, können dem Projekt Instanzen des Moduls über "Add New Item..." im Kontextmenü des Moduls hinzugefügt werden.

Debuggen:

TwinCAT-Module können sowohl in der lokalen als auch auf einer seperaten Runtime ausgeführt und debuggt werden.

Um lokal debuggen zu können muss zunächst eine Instanz des Moduls gestartet und verbunden werden. Dies kann direkt über das Menü des Moduls in Visual Studio erfolgen. Abbildung 6.7 zeigt, wie eine neue Modul-Instanz für das Debuggen gestartet werden kann.

Sollte bereits eine Modul-Instanz in einer Runtime vorhanden sein, muss diese zum Debuggen verbunden werden. Diese Verbindung kann über die Option "Attach to Process..." des "Debug"-Menüs in Visual Studio hergestellt werden. Abbildung 6.8 zeigt das Fenster, in dem laufende TwinCAT-Module ausgewählt und über den Button "Attach" verbunden werden können.

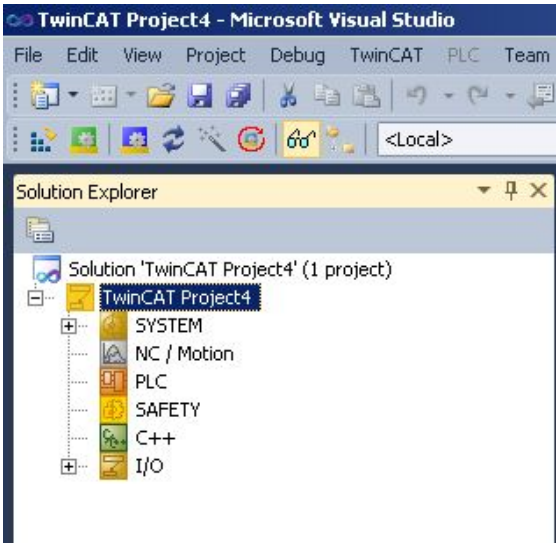


Abb. 6.6 Organisation eines Projektes[1]

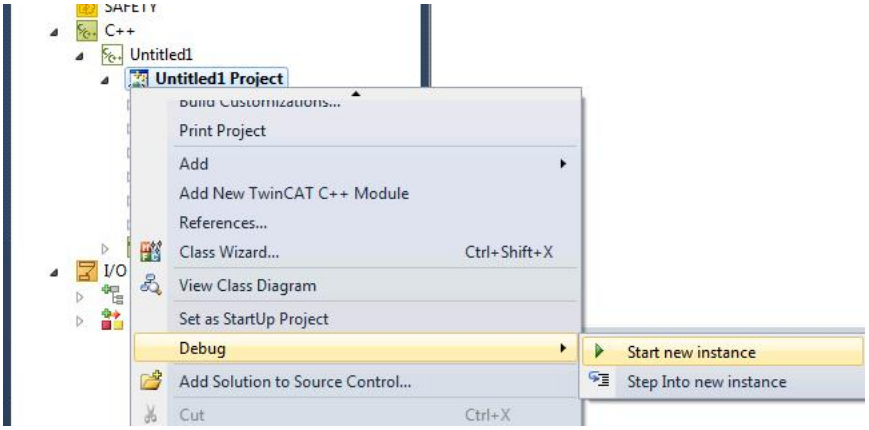


Abb. 6.7 Starten einer Debug-Instanz eines Moduls[1]

Nachdem ein Modul verbunden ist, können wie gewohnt Breakpoints gesetzt werden, an denen der Programmfluss angehalten wird. Solche Unterbrechungen sind in Automationsprozessen meist nicht erwünscht, da je nach Beschaffenheit des Systems Fehler auftreten können und eventuell das System beschädigt werden könnte. Um die Überwachung des Systemzustands, die zum Debuggen benötigt wird, dennoch zu ermöglichen stellt die "TwinCAT Live Watch". Diese erlaubt das Beobachten und die Manipulation von Variablen zur Laufzeit des Prozesses.

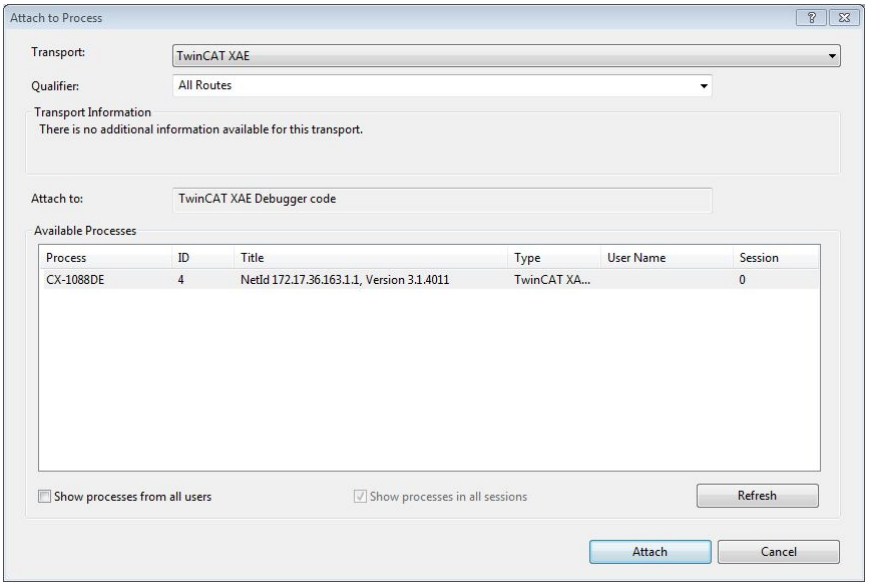


Abb. 6.8 Verbinden einer laufenden Modul-Instanz[1]

Am Ende des Debug-Vorgangs wird die Verbindung zur Modul-Instanz über die Option "Detach All" des "Debug"-Menüs wieder geschlossen.

6.5 Real-Time Ethernet

Feldbus-Systeme werden benötigt, um die Kommunikation zwischen dem Kontrollsystem und den Feldgeräten zu realisieren. Für PC-basierte Automation werden für die Verwendung von herkömmlichen Feldbustechniken spezielle Hardwaremodule benötigt. PCs besitzen typischerweise einen Ethernet-Anschluss, über den viele PCs verbunden werden können. Da PCs bereits über einen Ethernet-Anschluss verfügen, ist es naheliegend, Ethernet als Alternative zu herkömmlichen Feldbussystemen zu verwenden.

Automationsanlagen müssen dabei meist Realzeitanforderungen erfüllen. Realzeitanforderungen sind strenge zeitliche Vorgaben, die vom System eingehalten werden müssen, damit der korrekte Ablauf garantiert werden kann.
Ethernet selbst besitzt in seiner Standard-Form jedoch keine Mechanismen, die garantieren, dass solche Anforderungen nicht beeinflusst werden. Damit Realzeitanforderungen trotzdem erfüllt werden können, bedarf es einer Realzeit-Ethernet (RTE) Lösung. Abbildung 6.9 zeigt mögliche Strukturen, in denen RTE realisiert werden

kann. Im Wesentlichen bestehen drei mögliche Strukturen, um Realzeitkommunikation über Ethernet zu ermöglichen:

- Top of TCP/IP: Hier werden sämtliche Modifikationen für die Realzeitkommunikation in der Schicht über den standardisierten Ethernet-Protokollen vorgenommen.
- Top of Ethernet: RTE Lösungen mit diesem Aufbau umgehen/modifizieren die Standard-Ethernet-Protokolle
- Modified Ethernet: Neben der Modifikation der Standardprotokolle modifiziert diese Struktur auch das Ethernet selbst, um bessere Realzeitanforderungen zu garantieren.

Die Kommunikationszeiten, die benötigt werden, damit die Realzeitanforderungen nicht beeinträchtigt werden, lassen sich dabei in drei beispielhafte Klassen einteilen. [3]

- Die schwächste Klasse ist Human Control, bei der Kommunikationszeiten von ca. 100 ms benötigt werden. Diese Zeitanforderung wird typischerweise in Systemen benötigt, in denen menschlicher Einfluss Teil der Prozess- bzw. Systemüberwachung ist. RTE Lösungen, die diese Klasse erfüllen, können bereits mit einer Top of TCP/IP Struktur realisiert werden.
- Die zweite Klasse ist Process Control. Hier werden Kommunikationszeiten von weniger als 10 ms gefordert. Diese Klasse wird von Systemen benötigt, in denen PLCs oder PCs die Steuerung des Systems übernehmen. Für die Realisierung der Process Control Klasse müssen RTE Lösungen über eine gute Leistung verfügen, um das TCP/IP Protokoll in Realzeit ausführen zu können, oder eine Top of Ethernet Struktur besitzen, in der die Standardprotokolle durch Realzeit-Protokolle ausgetauscht werden.
- Die letzte Klasse ist Motion Control. Systeme mit sehr strengen Realzeitanforderungen, wie zum Beispiel für die synchronisierte Bewegung von mehreren Motoren, werden von dieser Klasse abgedeckt. Solche Systeme benötigen neben Kommunikationszeiten von maximal 1 ms auch eine zeitliche Abweichung von weniger als 1 μs zwischen den einzelnen Kommunikationen. Um diese zeitlichen Anforderungen zuverlässig erfüllen zu können, muss neben den Ethernet-Protokollen auch das Ethernet selbst modifiziert werden.

6.5.1 EtherCAT

Eine bereits etabilierte RTE Lösung ist EtherCAT. Diese RTE Lösung modifiziert das eigentliche Ethernet, um Kommunikationszeiten der Motion Control Klasse zu erreichen.

Die Kommunikation beruht auf dem Master/Slave Prinzip. Der Master, meist das Kontrollsystem, sendet Ethernet-Frames an die Slave-Knoten. Jedem Slave-Knoten

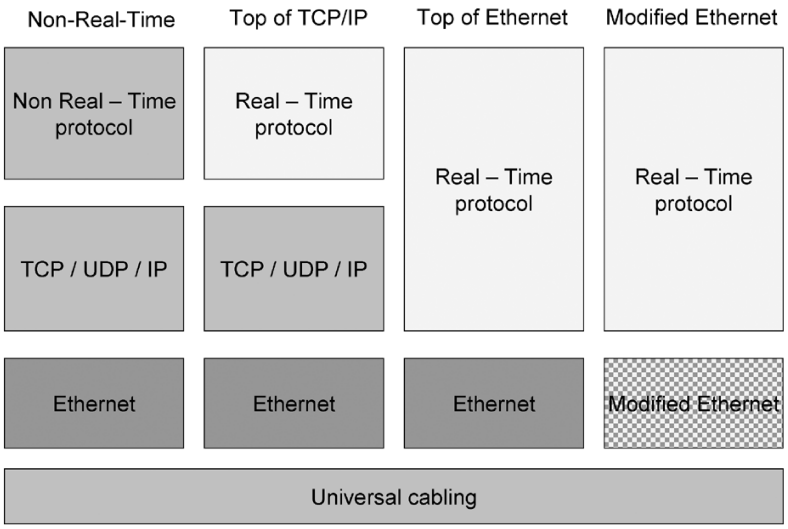


Abb. 6.9 Strukturen für RTE[3, S.1121]

(Feldgerät) ist dabei ein bestimmter Bereich des Frames zugeordnet, in dem die für das Gerät relevanten Daten gespeichert sind. In dem Slave-Knoten werden diese Daten mit den Informationen für das Kontrollsystem ausgetauscht. Ein Ethernet-Frame wird dabei in einer bestimmten Reihenfolge von den Slave-Knoten zu dem jeweils nächsten weitergeleitet. Der letzte Knoten in der Reihefolge sendet das Frame, in dem nur noch die Informationen für das Kontrollsystem stehen, an den Master als Antwort zurück.

Die Bearbeitung der Ethernet-Frames in den einzelnen Slave-Knoten würde zu Verzögerungen in jedem Knoten führen. Um diese Verzögerungen zu minimieren werden in den Slave-Knoten modifizierte Ethernet-Controller eingesetzt. Diese Controller entnehmen und ersetzen den entsprechenden Datenbereich des Ethernet-Frames, während es bereits an den nächsten Knoten weitergeleitet wird (Abbildung 6.10) [6].

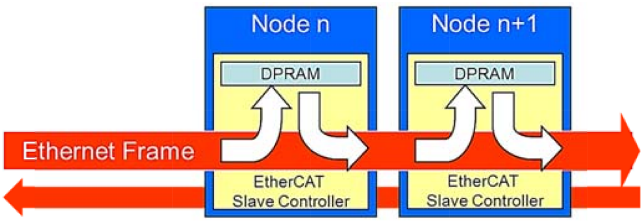


Abb. 6.10 ¨on the fly¨ Frame Processing[6, S.3]

Durch diese Anpassungen erlaubt EtherCAT Kommunikationszeiten von ca. 30-100 μs, je nachdem, ob in dem EtherCAT-Netwerk auch nicht-Realzeit-Kommunikationen stattfinden.

Für die Synchonisation von räumlich verteilten Prozessen verwendet EtherCAT verteilte Uhren. Die Synchonisation dieser Uhren erfolgt über die exakte Ermittlung des Laufzeitversatzes zwischen der Mutter-Uhr und den einzelnen Tochter-Uhren. [5]

Profinet CBA

Neben EtherCAT existeren weitere RTE Lösungen. Eine dieser Lösungen ist Profinet CBA (component-based automation). Profinet CBA nutzt Remote Procedure Call (RPC) und das Distributed Component Object Model (DCOM) Protokoll über eine standard TCP/IP Verbindung um Kommunikationszeiten der Human Control Klasse zu erreichen. Sollten bessere Kommunikationszeiten benötigt werden, umgeht Profinet CBA das TCP/IP Protokoll und ersetzt es durch ein Realzeit-Protokoll. Das Realzeit-Protokoll verwendet einen speziellen Ethertype mit Frame Priorisierung. Durch die bevorzugte Behandlung von Realzeit-Frames werden Kommunikationszeiten von weniger als 10 ms erreicht. [3]

6.6 Zusammenfassung

Nachdem gezeigt wurde, dass die gegenwärtigen PC-basierten Systeme die Anforderungen an Automationssysteme heute auf gleichem Niveau wie PLC erfüllen können, bietet es sich an, das zur Verfügung gestellte PC-basierte Automationssystem für die Umsetzung des Projektes zu verwenden. Dies erlaubt es der Projektgruppe die bekannten Programmiersprachen und Strukturen für PCs im Projekt zu nutzen.

Aufgrund des Aufbaus der Automatisierungssuite TwinCAT, die sowohl eine Entwicklungsumgebung für Automationsprogramme als auch eine Runtime stellt, mit der PC-Systeme in Automationsanlagen gewandelt werden können, bietet TwinCAT ein breites Spektrum an Werkzeugen für die Umsetzung der Simulation innerhalb des Projekts. Da die PC-basierten Komponenten der Automationsanlage bereits über eine Impletmentation der Realzeit-Ethernet Lösung "EtherCAT" verfügen, bietet es sich an diese zu Verwenden.

Literaturverzeichnis

1. Beckhoff Automation GmbH: Beckhoff Information System. URL `http://infosys.beckhoff.de/index.php?content=../content/1031/tc3_c/index.html&id=13475`. Letzter Zugriff 01.06.2014

2. Beckhoff Automation GmbH: TwinCAT 3 | eXtended Automation (XA) (2012). URL http://download.beckhoff.com/download/document/catalog/Beckhoff_TwinCAT3_042012_e.pdf
3. Felser, M.: Real-Time Ethernet – Industry Prospective. pp. 1118–1128 (2005)
4. Lipson, P., van der Zalm, G.: Inside Machines: PC versus PLC: Comparing control options. Webseite. URL http://www.controleng.com/single-article/inside-machines-pc-versus-plc-comparing-control-options/9bf8690c6f23b11370bec90b52cb15c9.html. Letzter Zugriff 01.06.2014
5. Rostan, M.: Der Feldbus heißt EtherCAT (2004). URL ethercat.de/pdf/german/Der_Feldbus_heisst_EtherCAT.pdf
6. Rostan, M., Stubbs, J., Dzilno, D.: Ethercat enabled advanced control architecture. In: Advanced Semiconductor Manufacturing Conference (ASMC), 2010 IEEE/SEMI, pp. 39–44 (2010). DOI 10.1109/ASMC.2010.5551414

Kapitel 7
Lineare Optimierung

Kevin Möhlmann

Zusammenfassung Die folgende Ausarbeitung schildert die Problematik der linearen Programmierung und der dort behandelten linearen Programmen. Dabei legt sie Wert auf eine anschauliche Beschreibung des Sachverhalts sowie eine korrekte Darstellung des Formalismus. Anschließend werden die Eckpunkt-Berechnungsmethode sowie der Simplexalgorithmus eingeführt, die zur Lösung von linearen Programmen verwendet werden können. Später wird kurz auf die Dualität bei linearen Programmen eingegangen.

7.1 Einleitung

Diese Einführung in die lineare Programmierung soll der Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“ (kurz: PG) als Hilfestellung dienen und einen möglichen Lösungsansatz für die von der PG betrachteten Optimierungsprobleme geben. Die PG hat die Aufgabe Gebäude zu simulieren, die unabhängig von externen Energiequellen, wie dem Strom- und dem Gasnetz, agieren. Dabei werden eine Reihe von Energieproduzenten und Energieverbrauchern betrachtet. Die Energieproduzenten erzeugen elektrische Energie und/oder Wärmeenergie und unterliegen dabei Einschränkungen, wie z.B. Kapazitätsgrenzen. Die Energieverbraucher verbrauchen ihrerseits die erzeugte Energie, wobei der Bedarf ebenfalls nach unten und nach oben beschränkt ist. Das Ziel eines energieautonomen Gebäudes ist es dabei, zu jedem Zeitpunkt genügend Energiereserven kostengünstig bereit zu stellen. Dafür muss die Energie nicht nur erzeugt sondern auch zwischengespeichert werden. Die Aufgabe der Bereitstellung kostengünstiger Energie kann u.U. als lineares Optimierungsproblem formuliert werden.

Carl von Ossietzky Universität Oldenburg
E-mail: kevin.moehlmann@uni-oldenburg.de

7.2 Lineare Programmierung

Die lineare Programmierung hat wenig mit der Erstellung von Softwareprogrammen zu tun, wie der Begriff irrtümlich vorgibt. Vielmehr werden in der linearen Programmierung Optimierungsprobleme betrachtet, bei denen eine lineare Zielfunktion mit Hilfe einer Reihe linearer Nebenbedingungen maximiert bzw. minimiert werden soll. Daher hat sich im Deutschen auch der Begriff lineare Optimierung durchgesetzt. Der aus dem Englischen stammende Begriff lineare Programmierung kommt aus einer Zeit, in der es außer zu Forschungszwecken keine Computer gab. Das Wort Programmierung wurde vielmehr im Sinne von Planung verstanden. (vgl. S.1 [6], S.9 [4])

7.2.1 Lineares Optimierungsproblem

Zu Anschauungszwecken wird zunächst ein Beispiel für ein lineares Optimierungsproblem gegeben. Ein Gebäude mit verschieden Anlagen zur Erzeugung von elektrischer- und thermischer Energie kann überschüssige Energieeinheiten verkaufen. Dabei bringt der Erlös von einer Einheit elektrischer Energie x dreimal soviel wie der Erlös von einer Einheit thermischer Energie y. Dies lässt sich als Funktion ausdrücken:

$$f = 3x + y$$

Diese Funktion gilt es natürlich zu optimieren, da davon der Gewinn abhängt. Die Anlagen zur Erzeugung der Energie unterliegen allerdings Einschränkungen, die sich wie folgt als Nebenbedingungen über die Variablen x und y ausdrücken lassen.

$$x \geq 0, y \geq 0$$
$$2y - x \geq 2$$
$$x + 3y \geq 6$$

Allgemein besteht das lineare Optimierungsproblem aus einer linearen Zielfunktion, welche mit Hilfe einer Reihe von Nebenbedingungen maximiert bzw. minimiert werden soll. Diese Nebenbedingungen sind in der Form von Ungleichungen über denselben Variablen wie die Zielfunktion formuliert. Wir betrachten im Folgenden nur die Aufgabe der Maximierung einer Zielfunktion. Die Aufgabe der Minimierung gilt analog. Allgemein stellt sich das lineare Optimierungsproblem mit $x_i, c_i, a_{ji} \in R$ und $b_j \in R$ wie folgt dar:
Zielfunktion:

$$c_1 x_1 + ... + c_n x_n = max!$$

Nebenbedingungen:

$$a_{11} x_1 + ... + a_{1n} x_n \leq b_1$$
$$a_{21} x_1 + ... + a_{2n} x_n \leq b_2$$
$$a_{m1} x_1 + ... + a_{mn} x_n \leq b_m$$

Alternativ lässt sich das lineare Optimierungsproblem bzw. auch lineares Programm oder kurz LP genannt, als Matrix A $\in R^{m \times n}$ mit zwei Spaltenvektoren b $\in R^m$ und c $\in R^n$ formulieren. Dabei soll ein Spaltenvektor x $\in R^n$ bestimmt werden, der die Ungleichung $Ax \leq b$ erfüllt und das Skalarprodukt $c^T x$ maximiert oder aber entschieden werden, ob es keine zulässige Lösung gibt oder unbeschränkt viele. In solchen Fällen wird das Problem als unzulässig bzw. unbeschränkt bezeichnet. Die Schreibweise $a \leq b$ bedeutet hierbei für Vektoren der gleichen Dimension, dass die Ungleichung komponentenweise gilt. Eine Lösungen die alle Ungleichungen erfüllt und für $c^T x$ einen maximalen Wert liefert, wird als optimal bezeichnet. Ein lineares Programm wird oft auch in der Schreibweise $max\ \{\ c^T x : Ax \leq b\ \}$ oder einfach nur als $max\ \{\ cx : Ax \leq b\ \}$ angegeben. (vgl. S.9 & S. 57 [4], S.1ff [6], [2])
Für obiges Beispiel lautet die Matrix A$\in R^{2\times 2}$ und die zwei Spaltenvektoren b $\in R^2$ und c $\in R^2$:

A = $\begin{pmatrix} -1 & 2 \\ 1 & 3 \end{pmatrix}$ b = $\begin{pmatrix} 2 \\ 6 \end{pmatrix}$, c= $\begin{pmatrix} 3 \\ 1 \end{pmatrix}$ mit dem linearen Optimierungsproblem $max\ \{\ c^T x : Ax \leq b\ \}$.

Wie sich im nächsten Unterkapitel herausstellen wird, ist dieses Problem weder unzulässig noch unbeschränkt und es gibt eine Menge an zulässigen Lösungen sowie eine optimale Lösung.

Wir halten zudem fest (Beweis siehe S. 58 [4]):

„Ist eine LP weder unzulässig noch unbeschränkt, so besitzt es eine optimale Lösung“. S. 58 [4]

Im Folgenden wird gezeigt, wie das lineare Optimierungsproblem geometrisch interpretiert werden kann und wie dies zur Lösung eines LPs verwendet werden kann.

7.2.2 Geometrische Interpretation

Wird die Zielfunktion zunächst nicht betrachtet und ist das LP nicht unzulässig, so gibt es gar eine Menge an zulässigen Lösungen. Diese Menge ergibt sich „aus dem Schnitt endlich vieler Halbräume“ S. 59 [4].

Ein Teilmenge des R^n heißt Halbraum des R^n, wenn sie durch eine Hyperebene begrenzt wird. Eine Hyperebene ist ein Unterraum eines Vektorraums der Dimension n, wenn sie die Dimension n-1 hat. (vgl. S. 255 & S. 233 [2], S. 50 [6])

So sind Hyperebenen im R^2 durch Geraden und im R^3 durch Flächen gegeben. Eine Hyperebene teilt den Raum folglich in zwei Hälften (Halbräume). Im Falle der linearen Optimierung liegen dabei in der einen Hälfte die Menge der zulässigen Lösungen, während die andere Hälfte nur unzulässige Lösungspunkte beschreibt.

Bei einem LP werden die Hyperebenen durch die Ungleichungen bestimmt. Der Schnitt aller durch Hyperebenen gebildeten Halbräume legt die Lösungsmenge fest. Die entstehende Menge wird geometrisch durch einen Polyeder beschrieben.

„Ein Polyeder im R^n ist“ also „eine Menge vom Typ $P = \{x \in R^n : Ax \leq b\}$, wobei $A \in R^{m \times n}$ eine Matrix und $b \in R^m$ ein Vektor ist.“ S, 59 [4]

Für das obige Beispiel sind die Hyperebenen durch die beiden Ungleichungen $2y - x \leq 2$ und $x + 3y \leq 6$ und die Bedingungen $x \geq 0$ und $y \geq 0$ gegeben. Dies lässt sich in einem zweidimensionalen kartesischem Koordinatensystem darstellen. Die beiden Achsen und die durch die beiden Ungleichungen gegebenen Geraden legen dabei die Menge der zulässigen Lösungen fest. Abbildung 7.1 veranschaulicht dies.

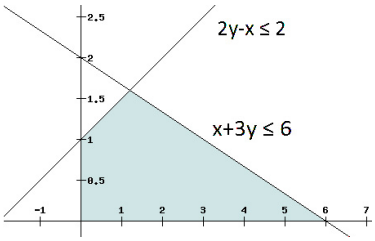


Abb. 7.1 grafische Darstellung eines LPs

Ein Polyeder wird als konvex bezeichnet, falls er für je zwei Punkte $x,y \in R^n$ auch alle Punkte enthält, die zwischen x und y auf einer Graden liegen, die durch x und y verläuft. Grundsätzlich sind alle Polyeder linearer Optimierungsprobleme konvex. Daher bildet die Menge aller zulässigen Lösungen eines LPs eine konvexe Punktmenge. (vgl. S. 48 [6], S.80f [9])

Ein Punkt einer konvexen Punktmenge kann ein Randpunkt oder ein innerer Punkt sein. Ein innerer Punkt ist ein Punkt, für den ein ϵ - Umgebung mit $\epsilon > 0$ angegeben werden kann, sodass alle Punkte innerhalb dieser Umgebung auch zur Punktmenge gehören. Kann solch eine Umgebung nicht angegeben werden, heißt der Punkt ein Randpunkt. Jeder Randpunkt eines Polyeders liegt damit auf einer der begrenzenden Hyperebenen. Besondere Randpunkte stellen die Eckpunkte dar. Eine Ecke bzw. ein Eckpunkt eines konvexen Polyeders kann als Spitze solch eines Polyeders verstanden werden. Zu einer Ecke E kann keine Strecke angegeben werden, die sowohl vollständig im Polyeder liegt, als auch E als Mittelpunkt hat. Formell lässt sich ein Eckpunkt als ein Punkt beschreiben, an der eine lineare Funktion ein Maximum annimmt. Ein Punkt v wird also als Ecke eines konvexen Polyeders $P \subset \in R^n$ bezeichnet, falls $v \in P$ und es einen Vektor $c \in R^n$ gibt, der nicht null ist, sodass $c^T v > c^T y$ für alle $y \in P \setminus \{v\}$ gilt. Ein Eckpunkt wird auch als Basislösung bezeichnet. (vgl. S. 53 [6], S. 82 [9], S.59 [4])

Es gilt:

In einem beschränkten und konvexen Polyeder erreicht eine lineare Funktion $f = c_1x_1 + c_2x_2 + ... + c_nx_n$ ihr Maximum in einem Eckpunkt. (vgl. S.82f [9])

Diese Erkenntnis ist für die zwei nachfolgenden Lösungsverfahren von linearen Programmen sehr wichtig. Deshalb folgt der Beweis dafür, wie er in vgl. S. 82ff [9] geführt wurde. Dabei wird zunächst gezeigt, dass die Funktion f ihren maximalen Wert nur in einem Randpunkt erreicht. Anschließend wird gezeigt, dass der optimale Wert nur in einem Eckpunkt gefunden werden kann, sofern nur einem Punkt der optimale Wert angenommen wird.
Seien $P'(x'_1, x'_2, \ldots, x'_n)$ und $P''(x''_1, x''_2, \ldots, x''_n)$ zwei Randpunkte des Polyeders. Die Funktion f nimmt für diese Punkte die folgenden Werte an:

$$f' = c_1x'_1 + c_2x'_2 + ... + c_nx'_n$$
$$f'' = c_1x''_1 + c_2x''_2 + ... + c_nx''_n$$

Ein Punkt $P(x_1, x_2, ..., x_n)$ auf einer von P' und P'' bestimmten Geraden g hat die Koordinaten

$$x_1 = kx''_1 + (1-k)x'_1$$
$$x_2 = kx''_2 + (1-k)x'_2$$
$$...$$
$$x_n = kx''_n + (1-k)x'_n \; wobei \; 0 \leq k \leq 1$$

Da der Polyeder konvex ist, sind alle Punkte P auf g auch in der Lösungsmenge enthalten. Es sind also zulässige Punkte.

Werden die Werte in die Funktion $f = c_1x_1 + c_2x_2 + ... + c_nx_n$ eingesetzt, so ergibt sich daraus

$$\begin{aligned} f &= c_1[kx''_1 + (1-k)x'_1] + c_2[kx''_2 + (1-k)x'_2] + ... + c_n[kx''_n + (1-k)x'_n] \\ &= k[c_1x''_1 + c_2x''_2 + ... + c_nx''_n] + (1-k)[c_1x'_1 + c_2x'_2 + ... + c_nx'_n] \\ &= kf'' + (1-k)f' \; wobei \; 0 \leq k \leq 1 \end{aligned}$$

Um dieses Ergebnis zu deuten, wird eine Fallunterscheidung getroffen:

1. $f'' = f'$
2. $f'' < f'$
3. $f'' > f'$

Aus dem ersten Fall folgt:

$$\begin{aligned} f &= kf'' + (1-k)f' && -kf'' \\ \Leftrightarrow f - kf'' &= f' - kf' && : kf' und f'' = f' \\ &\Leftrightarrow f = f' \end{aligned}$$

Die Funktion f nimmt also für f"=f' auf der gesamten Geraden g zwischen den Punkten P' und P" den gleichen Wert an.

Aus dem zweiten Fall folgt:

$$\begin{aligned} f = kf'' + (1-k)f' = kf'' + f' - kf' && - f' \\ \Leftrightarrow f - f' = k(f'' - f') \end{aligned}$$

Mit der Einschränkung $0 \leq k \leq 1$ und $f'' < f'$ folgt, dass $k(f'' - f')$ kleiner oder gleich null ist und damit $f - f' \leq 0$ bzw. $f \leq f'$ gilt. Die Funktion f erreicht damit ihr Maximum in dem Punkt P' mit dem Funktionswert f'.

Aus dem dritten Fall folgt:

Analog zum zweiten Fall folgt aus $0 \leq k \leq 1$ und $f'' > f'$, dass gilt $f \leq f''$. Die Funktion f erreicht also ihr Maximum in dem Punkt P" mit dem Funktionswert f".

Damit ist gezeigt, dass die Funktion f ihren maximalen Wert in einem Randpunkt annimmt. Die Randpunkte müssen also stets zum Definitionsbereich gehören, um Extremwerte und damit optimale Lösungen finden zu können. Dafür müssen die Nebenbedingungen tatsächlich von der Art $a_{11}x_1 + ... + a_{1n}x_n \leq b_1$ sein, also das Gleichheitszeichen mit einschließen, vorausgesetzt die Grundmenge ist die Menge der reellen Zahlen.

Sei P_m ein Punkt, für den die Funktion f ihren maximalen Wert in einem beschränkten und konvexen Polyeder R annimmt. Sei dies zudem der einzige Punkt für den f in R einen maximalen Wert annimmt. Dann ist dieser Punkt ein Eckpunkt. Wäre dies nicht der Fall, so gäbe es eine Strecke, die durch P_m verläuft und ausschließlich durch den Rand geht, sodass P_m kein Eckpunkt ist. Demnach aber würden sämtliche andere Punkte auf dieser Strecke einen Funktionswert annehmen, der kleiner oder gleich als der von P_m ist. Dies steht aber im Widerspruch zum Beweis von oben, dass f nur in einem äußeren Punkt der Strecke den maximalen Funktionswert annimmt bzw. zur Voraussetzung, dass f nur in P_m den maximalen Wert annimmt.

Mit dieser Beweisführung ist gezeigt, dass eine Zielfunktion für einen beschränkten und konvexen Polyeder ihren maximalen Wert in einem Eckpunkt erreicht. (vgl. S. 82ff [9])

7.3 Eckpunkt-Berechnungsmethode

Die Eckpunkt-Berechnungsmethode bestimmt für ein lineares Optimierungsproblem den optimalen Wert. Dabei macht sie sich die Tatsache zu Nutze, dass der optimale Wert in einem Eckpunkt des Polyeders zu finden ist, welcher die Menge der zulässigen Lösungen beschreibt.

Diese Methode der Lösung eines LPs ist zwar aus theoretischer Sicht nicht schwierig umzusetzen. Jedoch ist der Rechenaufwand für die Berechnung sämtlicher Eckpunkte so groß, dass die Eckpunkt-Berechnungsmethode zur Lösung von praktischen Aufgaben irrelevant ist. Denn zur Berechnung der Eckpunkte muss im n-dimensionalen Raum für jeweils n begrenzende Hyperebenen der Schnittpunkt ermittelt werden, bis alle Eckpunkte bekannt sind.

Sind die Eckpunkte bekannt, so muss zunächst überprüft werden, ob die gefunden Eckpunkte auch tatsächlich innerhalb des Polyeders liegen. D.h. es muss überprüft werden, ob alle einschränkenden Bedingungen erfüllt sind. Ist dies der Fall, so liegt ein zulässiger Eckpunkt vor. Anschließend werden die Eckpunkte als Funktionswerte in die Zielfunktion eingesetzt und berechnet. Selbst wenn der optimale Wert bereits berechnet wurde, werden die Funktionswerte aller anderen Eckpunkte ebenfalls bestimmt. Erst anschließend ist der größte Wert als maximale Lösung bestimmt. (vgl. S. 84ff [9])

Um für das Einführungsbeispiel den optimalen Wert bestimmen zu können, werden also zunächst alle Eckpunkte bestimmt, indem die Schnittpunkte aller Geraden berechnet werden.

Aus $2y - x \leq 2$ und $x + 3y \leq 6$ werden die Geradengleichung durch Umformung und Ersetzen der Ungleichung durch eine Gleichung gewonnen. Es ergibt sich $y = 1 + x/2$ und $y = 2 - x/3$. Die Schnittpunkte der Geraden miteinander sowie mit den Achsen sind in der folgenden Tabelle aufgelistet.

Gleichung zur Bestimmung der Schnittpunkte	Wertepaare der Schnittpunkte (x,y)	Wert der Zielfunktion $f = 3x + y$	Ist der Eckpunkt zulässig?
$1 + x/2 = 2 - x/3$	$(6/5, 8/5)$	5,2	Ja
$0 = 1 + x/2$	$(-2, 0)$	Nicht relevant	Nein
$y = 1 + 0/2$	$(0, 1)$	1	Ja
$0 = 2 - x/3$	$(6, 0)$	18	Ja
$y = 2 - 0/3$	$(0, 2)$	Nicht relevant	Nein

Nur für zulässige Eckpunkte wurden die Funktionswerte bestimmt. Der größte Wert beträgt 18 und wird an der Stelle x=6 und y=0. Dieser Wert ist die optimale Lösung des linearen Programms.

7.4 Simplexalgorithmus

Eine weitaus bessere Möglichkeit zur Lösung von linearen Optimierungsproblemen als die Eckpunkt-Berechnungsmethode stellt der Simplexalgorithmus dar. Die-

ses von Dantzig 1947 veröffentlichte algebraische Iterationsverfahren kann ein LP mit beliebig vielen Variablen lösen. Dabei nutzt es ebenfalls wie die Eckpunkt-Berechnungsmethode den Sachverhalt aus, dass sich die optimale Lösung hinter einem Funktionswert eines Eckpunkts verbirgt. Ausgehend von einem beliebigen bereits bekannten zulässigen Eckpunkt werden nach dem Simplexalgorithmus neue Eckpunkte berechnet, um schrittweise eine Verbesserung der Zielfunktion zu erreichen. Dies erfolgt so lange, bis die optimale Lösung gefunden ist. (vgl. S.88 [9], S.63 [4])

Für den Simplexalgorithmus werden die Ungleichungen der Nebenbedingungen durch Schlupfvariablen in Gleichungen überführt. Im Rahmen der Einführung wird an dieser Stelle zunächst vorausgesetzt, dass für alle b_j und alle x_i gilt: $b_j > 0$ und $x_i \geq 0$. Später wird aufgezeigt, wie das Verfahren auf negative b_j bzw. negative x_i anzuwenden ist.

Die Einführung von Schlupfvariablen $s_i \geq 0$ führt zu folgendem System S_1:

$$\begin{aligned} a_{11}x_1 + ... + a_{1n}x_n + s_1 &= b_1 \\ a_{21}x_1 + ... + a_{2n}x_n + s_1 &= b_2 \\ &... \\ a_{m1}x_1 + ... + a_{mn}x_n + s_1 &= b_m \end{aligned}$$

Es gibt also m Gleichungen mit m+n Variablen. Da aus m Gleichungen m Variablen bestimmt werden können, werden alle x_i zunächst auf null gesetzt. Dies führt zur ersten Basislösung

$$s_1 = b_1, s_2 = b_2, ..., s_m = b_m$$

mit dem Funktionswert $f = c_1x_1 + ... + c_nx_n = 0$.

Die auf null gesetzten Variablen werden Nichtbasisvariablen genannt. Die nicht auf null gesetzten Variablen werden als Basisvariablen bezeichnet.

Ausgehend von dieser Basislösung, welche einen Eckpunkt im Polyeder des LPs beschreibt, soll schritt-weise eine bessere Lösung gefunden werden. Da alle x_i die Nichtnegativitätsbedingung erfüllen müssen, also größer als null sein müssen, kann sich der Wert der Funktion f durch Finden einer anderen Basislösung noch verbessern, sofern f positive Koeffizienten c_i besitzt.

Die Suche nach einer besseren Basislösung wird beim Simplexalgorithmus mit Hilfe eines Simplextableaus vollzogen. Bis auf die erste Zeile sowie die erste und letzte Spalte stellt das Simplextableau die Koeffizienten des Systems S_1 sowie die Koeffizienten der Zielfunktion als Matrix dar, bei dem initial die Koeffizienten der Schlupfvariablen auf eins gesetzt wurden. Die erste Zeile dient nur der Übersicht. In ihr werden die zu den Koeffizienten gehörenden Variablen aufgetragen. In der ersten Spalte werden ebenfalls zu Übersichtszwecken die Namen der Basisvariablen vermerkt. In der letzten Spalte stehen die Koeffizienten des Lösungsvektors b. Das

initiale Simplextableau sieht also wie folgt aus:

Basisvariable	x_1	...	x_n	s_1	...	s_m	b_j
s_1	a_{11}		a_{1n}	1		0	b_1
...		...			...		...
s_m	a_{m1}		a_{mn}	0		1	b_m
Ziel	c_1		c_n	0		0	$0 = max$

Für obiges Beispiel ergibt sich das folgende System von Gleichungen und folgendes Simplextableau:

$$x \geq 0, y \geq 0 \; und \; s_1 \geq 0, s_2 \geq 0$$
$$2y - x + s_1 = 2$$
$$x + 3y + s_2 = 6$$
$$f = 3x + y$$

Basisvariable	x	y	s_1	s_2	b_j
s_1	-1	2	1	0	2
s_2	1	3	0	1	6
Ziel	3	1	0	0	0

Um nun zu einer besseren Basislösung zu gelangen, wird eine Basisvariable gegen eine Nichtbasisvariable getauscht. Dabei wird die Nichtbasisvariable gewählt, bei der die Zielfunktion den größten zugehörigen Koeffizienten $c_i > 0$ annimmt. Die Spalte, in der die Nichtbasisvariable steht, wird als Pivotspalte oder Hauptspalte bezeichnet. Ist kein Koeffizient $c_i > 0$ vorhanden, so terminiert das Verfahren. Die bestmögliche Lösung ist gefunden.

Die Basisvariable, die zum Austausch gewählt wird, muss für die gewählte Pivotspalte k folgende Bedingung erfüllen:

$$\frac{b_j}{a_{jk}} = min_{i \in \{1,\ldots,m\}} \{ \frac{b_i}{a_{ik}} \; : \; \frac{b_i}{a_{ik}} > 0 \}$$

Die so bestimmte Zeile j wird Pivotzeile oder Hauptspalte genannt. Das Element a_{jk}, das sich an der Stelle befindet, wo sich Pivotzeile und Pivotspalte kreuzen, heißt Pivotelement oder Hauptelement.

Nach der Bestimmung des Pivotelements findet der eigentliche Austausch der Basisvariablen gegen die Nichtbasisvariable statt. Dazu wird ein neues Simplextableau erstellt, das aufgrund der Wahl des Pivotelements auch die Bedingung $b_j > 0$ für alle b_j erfüllt. Dafür wird zunächst die Pivotzeile j durch das Pivotelement a_{jk} dividiert, sodass das Pivotelement den Wert eins bekommt. Aus allen anderen Zeilen wird die zum Pivotelement a_{jk} gehörende Variable x_k eliminiert. D.h. die dazugehörigen Koeffizienten erhalten den Wert null. Die zum Pivotelement gehörende Variable wird folglich nur noch durch eine Zeile bzw. eine Gleichung der Matrix beschrieben. Die Eliminierung von x_k geschieht für jede Zeile $i \in \{1, \ldots, m\}$ mit $i \neq j$ und jeder betrachteten Spalte $l \in \{1, \ldots, n\}$ auf folgende Weise:

$$a'_{il} = a_{il} - a_{ik}a_{jl}/a_{jk}$$
$$b'_i = b_i - a_{ik}b_j/a_{jk}$$
$$c'_l = c_l - c_k a_{jl}/a_{jk}$$
$$max' = max - c_k b_j/a_{jk}$$

Von jeder Zeile i wird also a_{ik}-fache der normierten Pivotzeile bzw. im Falle der Zielzeile das c_k-fache der normierten Pivotzeile abgezogen. Die Variable x_k hängt nach diesem Verfahren nur noch von einer Gleichung der Matrix ab. Damit ist x_k zur neuen Basisvariablen geworden. Der zu x_k gehörende Koeffizient hat den Wert eins.

Dieser Vorgang eines Tausches einer Basisvariablen gegen eine Nichtbasisvariable wird so lange wiederholt, bis die Zielzeile des Simplextableaus keine positiven Koeffizienten mehr enthält.
Diese Abbruchbedingung wird als Simplex-Kriterium bezeichnet:

„Eine Basislösung ist nicht maximal, wenn die Zielfunktion in der Basisdarstellung noch positive Koeffizienten hat.“ (S.103 [9])

Gibt es noch einen positiven Koeffizienten in der Zielfunktion, aber es kann kein Element b_j als Pivotelement bestimmt werden, da b_j/a_{jk} für alle möglichen Kandidaten kleiner als null ist, so gibt es keine endliche Lösung für das betrachtete LP. Der Simplexalgorithmus terminiert.

Für das Beispiel ergibt sich folgender Schritt:

Der größte Koeffizient der Zielfunktion ist 3 und liegt in der ersten Spalte. Damit ist die erste Spalte die Pivotspalte und x die auszutauschende Nichtbasisvariable. Es gilt:

$$\{\tfrac{b_i}{a_{i1}} \ : \ \tfrac{b_i}{a_{i1}} > 0\} = min\{\tfrac{b_2}{a_2 1}\} = \tfrac{b_2}{a_{21}} = 6$$

Damit ist die zweite Zeile die Pivotzeile und s_2 die auszutauschende Basisvariable. Das Element a_{21} ist als Pivotelement bestimmt. Jeder Eintrag der Pivotzeile wird durch das Pivotelement dividiert. Anschließend wird von der ersten Zeile das -1-fache der Pivotzeile und das 3-fache der Pivotzeile von der Zielzeile abgezogen. Es ergibt sich das folgende Simplextableau:

Basisvariable	x	y	s_1	s_2	b_j
s_1	0	5	1	1	2
x	1	3	0	1	6
Ziel	0	-8	0	-3	-18

Da keine positiven Koeffizienten in der Zielzeile mehr auftauchen, terminiert der Algorithmus.

Aus dem Simplextableau lässt sich ablesen, welche Variablen Basisvariablen sind und welche Werte diese annehmen. Die Basisvariablen sind im Beispiel durch x und s_1 gegeben, wie in der linken Spalte zu erkennen ist. Diese nehmen die Werte sechs und acht an, was grundsätzlich aus der rechten Spalte abgelesen werden kann. Alle anderen Variablen sind Nichtbasisvariablen, die keinen Einfluss auf die Basislösung haben. Sie haben dementsprechend den Wert null. In der letzten Spalte der Zielzeile ist zudem der negierte Wert der Basislösung abzulesen. Im Beispiel ist dies der Wert -18. D.h. die optimale Lösung des LPs beträgt 18 und wird mit x=6 und y=0 erreicht. (vgl. [3], [9])

7.4.1 Umgang mit Gleichungen

Bisher wurden nur Fälle betrachtet, bei denen die Nebenbedingungen als Ungleichungen formuliert sind. Nun soll der Fall kurz skizziert werden, dass eine Gleichung unter den Nebenbedingungen auftritt. Wir betrachten also Nebenbedingungen der folgenden Art:

$$a_{11}x_1 + ... + a_{1n}x_n = b_1$$

Damit ein LP mit einer Gleichung als Nebenbedingung mit dem Simplexverfahren gelöst werden kann, wird wie beim Simplexverfahren üblich eine Schlupfvariable eingeführt. Für jede Gleichung wird diese Schlupfvariable gekennzeichnet.

$$a_{11}x_1 + ... + a_{1n}x_n + s'_1 = b_1$$

Dabei darf jede mit einem Strich gekennzeichnete Variable s'_i in der Lösung nur den Wert null erhalten, da sonst die Gleichung nicht erfüllt wäre. Um dies zu bewerkstelligen, wird eine weitere Zielfunktion für jede Gleichung unter Nebenbedingungen eingeführt. Diese sekundäre Zielfunktion hat dieselben Koeffizienten und Variablen wie die Gleichung, für die sie eingeführt wurde. Es gilt dabei, den Funktionswert zu maximieren.

$$a_{11}x_1 + ... + a_{1n}x_n = max!$$

Die sekundäre Zielfunktion wird zunächst unter Beachtung aller Nebenbedingungen nach dem Simplexverfahren maximiert. Wenn das gefundene Maximum dabei einen Wert $< b_1$ annimmt, so gibt es keine Lösung für das LP. Nimmt das Maximum genau den Wert b1 an, so gibt es eine optimale Lösung für das LP und der Simplexalgorithmus liefert eine erste Ausgangslösung. Diese Ausgangslösung wird anschließend noch mit der primären Zielfunktion maximiert. Dabei muss allerdings

darauf geachtet werden, dass keine mit einem Strich gekennzeichnete Variable in der Basislösung auftaucht. (vgl. S.111ff [9])

Zur Verdeutlichung folgt ein Beispiel. Die linke Seite zeigt das Optimierungsproblem. Auf der rechten Seite sind die Schlupfvariablen und die sekundäre Zielfunktion bereits eingeführt.

1.	$x \geq 0, y \geq 0$	$x \geq 0, y \geq 0, \quad s_1 \geq 0, s_2 \geq 0, s_3' \geq 0$
2.	$2y \leq 2$	$2y + s_1 = 2$
	$x + 3y \leq 6$	$x + 3y + s_2 = 6$
	$2x + y = 2$	$2x + y + s_3' = 2$
3.	$x + y = max!$	$x + y = max!$
4.	$2x + y = 2$	(sekundäre Zielfunktion)

Es ergibt sich das unten stehende Simplextableau. Dabei geht die sekundäre Zielfunktion mit dem Funktionswert zwei in das Tableau mit ein. Das ist sinnvoll, da sich der Funktionswert bei jedem Schritt des Simplexalgorithmus verringert, sofern dieser Schritt eine Maximierung des Zielwerts bewirkt. Dieser Wert muss schließlich den Wert null annehmen, denn dann ist das Maximum der sekundären Zielfunktion erreicht und die primäre Zielfunktion kann optimiert werden. Kann nicht der Wert null erreicht werden, so gibt es keine Lösung.

Basisvariable	x	y	s_1	s_2	s_3'	b_j
s_1	0	2	1	0	0	2
s_2	1	3	0	1	0	6
s_3	2	1	0	0	1	2
prim. Zielfkt.	1	1	0	0	0	0
sec. Zielfkt.	2	1	0	0	0	2

Das Pivotelement ist hervorgehoben. Nach dem Austauschschritt ist die sekundäre Zielfunktion bereits optimiert. Der Wert der sekundären Zielfunktion beträgt null.

Basisvariable	x	y	s_1	s_2	s_3'	b_j
s_1	0	2	1	0	0	2
s_2	0	5/2	0	1	-1/2	5
x	1	1/2	0	0	1/2	1
prim. Zielfkt.	0	1/2	0	0	-1/2	-1
sec. Zielfkt.	0	0	0	0	-1	0

Im nächsten Tableau wird die sekundäre Zielfunktion nicht mehr mit aufgeführt, da sie bereits optimiert ist. Es wird nur noch die primäre Zielfunktion betrachtet. Außerdem werden alle gekennzeichneten Schlupfvariablen nicht mehr betrachtet, da diese in der Basislösung den Wert null annehmen müssen.

Basisvariable	x	y	s_1	s_2	s_3'	b_j
y	0	1	1/2	0		1
s_2	0	0	5/4	1		5/2
x	1	0	−1/4	0		1/2
prim. Zielfkt.	0	0	−1/4	0		-3/2

Die gefundene Lösung wird durch x=1/2 und y=1 beschrieben und nimmt den Funktionswert $f = (-3)/2$ an. Sie ist optimal und berücksichtigt auch die als Gleichung formulierte Nebenbedingung. Folgende Grafik gibt dies wieder.

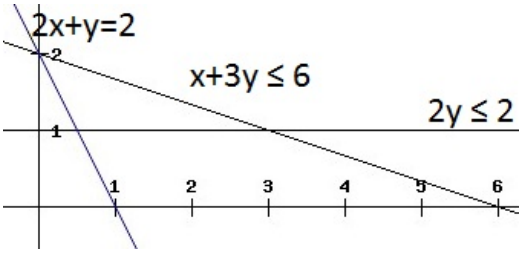


Abb. 7.2 Gleichung als Nebenbedingung

Da eine Nebenbedingungen als Gleichung auftritt, muss die optimale Lösung auf einem Schnittpunkt der Geraden von $2x + y = 2$ liegen. Es kommen tatsächlich nur zwei Eckpunkte in Frage.

7.4.2 Umgang mit Ungleichungen der Art T1 ≥ T2

Nachdem nun auch Gleichungen als Nebenbedingungen auftauchen dürfen, sollen jetzt Ungleichung der folgenden Art betrachtet werden:

$$a_{11}x_1 + ... + a_{1n}x_n \geq b_1$$

In diesem Fall wird durch Einführung einer Schlupfvariablen die Ungleichung in eine Gleichung umgeformt. Es ergibt sich:

$$a_{11}x_1 + ... + a_{1n}x_n - s_1 = b_1$$

Der einzige Unterschied zu den bisher betrachteten linearen Optimierungsaufgaben ist das Vorzeichen der eingeführten Schlupfvariablen. Diese Aufgabenstellung kann ebenfalls mit Hilfe einer sekundären Zielfunktion gelöst werden. Die sekundäre Zielfunktion darf dabei keinen Wert $< b_1$ annehmen, andernfalls hat das LP keine Lösung. (vgl. S.116ff [9])

7.4.3 Umgang mit negativen b_i

Treten Ungleichungen auf, bei denen auf der rechten Seite negative Zahlen auftauchen, so ist es möglich das Vorzeichen durch Multiplikation mit -1 zu drehen.

$$\begin{array}{ll} a_{11}x_1 + ... + a_{1n}x_n \geq -b_1 & *(-1) \\ \Leftrightarrow a_{11}x_1 + ... + a_{1n}x_n \leq b_1 & \end{array}$$

Dabei ändert sich allerdings auch die Richtung der Ungleichung. Das so erhaltene System muss folglich mit Hilfe einer sekundären Zielfunktion, wie es im vorangegangenen Absatz geschildert wurde, gelöst werden. (vgl. S.118 [9])

7.4.4 Minimum einer Zielfunktion bestimmen

Bisher wurde nur der Fall betrachtet, das Maximum einer Funktion zu bestimmen. Lautet die Opti-mierungsaufgabe allerdings wie folgt:

$$f = c_1x_1 + ... + c_nx_n = min!$$

So kann anstelle der Bestimmung des Minimums von f auch das Maximum von $\breve{}f$ bestimmt werden. Damit lautet die analoge Optimierungsaufgabe:

$$-f = c_1x_1 - ... - c_nx_n = max!$$

Auf diese Weise kann jedes LP behandelt werden, bei dem das Minimum bestimmt werden soll. (vgl. S.118f [9])

7.4.5 NichtN̄egativitätsbedingung sind aufgehoben

Es soll noch der Fall betrachtet werden, dass einzelne x_i auch negative Werte annehmen können. Sei im Folgenden $x_1, x_2, ..., x_{n-1} \geq 0$ und x_n beliebig mit:

$$\begin{array}{c} a_{11}x_1 + ... + a_{1n}x_n \leq b_1 \\ ... \\ \hline f = c_1x_1 + ... + c_nx_n = max! \end{array}$$

Für die Variable x_n wird dann eine neue Gleichung eingeführt mit zwei Variablen y_1, y_2. Und zwar wie folgt:

$$x_n = y_1 - y_2$$

Diese Gleichung wird für jedes Vorkommen von x_n in das bereits vorhandene System eingesetzt. Es ergibt sich:

$$\begin{gathered} x_1, x_2, ... \geq 0, \quad y_1, y_2 \geq 0 \\ a_{11}x_1 + ... + a_{1n}y_1 - a_{1n}y_2 \leq b_1 \\ ... \\ \hline c_1 + ... + c_n y_1 - c_n y_2 = max! \end{gathered}$$

Nach dieser Umformung müssen alle Variablen wieder die NichtÑegativitätsbedingung erfüllen insbesondere auch y_1 und y_2. Das Optimierungsproblem kann also in gewohnter Weise mit dem Simplexalgorithmus gelöst werden. (vgl. S.119f [9])

7.4.6 Laufzeit des Simplexalgorithmus

Die Laufzeit des Simplexalgorithmus ist abhängig von der Anzahl der besuchten Ecken des Lösungspolyeders. Für den allgemeinen Simplexalgorithmus haben Klee und Minty Beispiele angegeben, für die der Simplexalgorithmus exponentielle Laufzeit benötigt. In der praktischen Anwendung wird aber in den meisten Fällen nur ein linearer bis quadratischer Aufwand beobachtet. (vgl. [3])

7.5 Dualität bei lineare Programmen

Für die Einführung in die Dualität bei linearen Programmen soll zunächst ein Beispiel gegeben werden. Zur Herstellung zweier Geräte A und B ergibt sich folgendes LP:

	$x_1 \geq 0, x_2 \geq 0$	
Nebenbedingungen:	$4x_1 + x_2 \leq 2000$	(Gesamtkapazität der Fabrik zur Fertigung beider Geräte)
	$x_1 \leq 1000$	(Kapazität von Abteilung A zur Fertigung von Gerät A)
	$x_2 \leq 1500$	(Kapazität von Abteilung B zur Fertigung von Gerät B)
Zielfunktion:	$100x_1 + 90x_2 = max!$	

Dieses Problem lässt sich auch aus einer anderen Sicht betrachten. Die beiden Abteilungen zur Herstellung der Geräte A und B sowie die Gesamtfabrik unterliegen Einschränkungen. Diese Einschränkungen lassen sich durch Arbeitswerte a_A, a_B und a_{Ges} benennen. Die Arbeitswerte beschreiben dabei den Aufwand zur Herstellung von einem Gerät A bzw. einem Gerät B wie folgt:

a_{Ges} ist der Aufwand zur Herstellung von einem Gerät A oder B
a_A ist der Aufwand zur Herstellung von einem Gerät A
a_B ist der Aufwand zur Herstellung von einem Gerät B

Ein Arbeitnehmer wird stets versuchen, die Arbeitswerte klein zu halten. Hinter den Arbeitswerten verbergen sich schließlich Lohn- und Arbeitskosten. Es ergibt sich also folgende Zielfunktion:

$$2000a_{Ges} + 1000a_A + 1500a_B = min!$$

Für die Herstellung eines Gerätes A und B sind dabei aus der Problemstellung folgende Nebenbedingungen erkennbar:

$4a_{Ges} + 1a_A + 0a_B \leq 100$ (Arbeitswerte zur Herstellung von Gerät A)
$1a_{Ges} + 0a_A + 1a_B \leq 90$ (Arbeitswerte zur Herstellung von Gerät B)

Jeder Arbeitswert unterliegt natürlich der NichtÑegativitätsbedingung. Damit ergibt sich folgendes LP:

$$a_B \geq 0, a_A \geq 0, a_{Ges} \geq 0$$
$$4a_{Ges}s + 1a_A + 0a_B \geq 100$$
$$1a_{Ges} + 0a_A + 1a_B \geq 90$$
$$2000a_A + 1000a_B + 1500a_{Ges} = min!$$

Beide Probleme sind eng miteinander verknüpft. Die folgenden Matrizen, die die Koeffizienten beider LPs beinhalten, verdeutlichen dies:

x_1	x_2	b_j
4	1	2000
1	0	1000
0	1	1500
100	90	max

a_{GES}	a_A	a_B	b_j
4	1	0	100
1	0	1	90
2000	1000	1500	min

Bei Betrachtung beider Matrizen fällt auf, dass aus dem ursprünglichen LP das neue LP durch Vertauschen von Spalten mit Zeilen entsteht. Zudem sind die Relationen der einschränkenden Bedingungen und die Zielfunktion vertauscht. Werden beide LPs gelöst, so sind die Werte der beiden Zielfunktion, die durch Einsetzen

der Lösung berechnet werden kann, identisch. Die beiden Optimierungsprobleme werden als dual bezeichnet. Es gilt die Definition:

„Ist $max\ \{\ cx : Ax \leq b\ \}$ ein LP, so definieren wir das dazu duale LP als das lineare Programm $min\ \{yb : yA = c, y \geq 0\ \}$." (S.70 [4])

Damit ist jedem Maximumproblem ein duales Minimumproblem zugeordnet. Das ursprüngliche LP wird dabei als primales LP bezeichnet. Sind x und y beide optimale Lösungen, so gilt äquivalent $cx = yb$. Weiter gilt: „Ist ein LP unbeschränkt, so ist das dazu duale LP unzulässig. Hat ein LP eine optimale Lösung, so hat das dazu duale LP auch eine optimale Lösung." (S.73 [4])

Da die optimalen Lösungen cx und yb identisch sind, ist es gleichgültig welche Aufgabe mit dem Simplexalgorithmus gelöst wird. Oft ist es einfacher, zu einem LP das duale LP zu bestimmen und dieses zu berechnen. (vgl. [4], [9])

7.6 Hinweise zur Implementierung des Simplexalgorithmus

Nach der Bestimmung eines Pivotelements a_{ij} wird die gesamte Pivotzeile durch dieses dividiert. Anschließend wird von jeder Zeile a_{kj} mit $k \neq j$ das $a_{ij}-$fache abgezogen. Aufgrund der dort stattfindenden Divisionen und Multiplikationen sind sich fortpflanzende Rundungsfehler ein mögliches Problem des Simplexalgorithmus. Bei der Implementierung des Simplexalgorithmus muss daher darauf geachtet werden, dass der Algorithmus numerisch stabil abläuft. Die beschriebene Wahl des Pivotelements stellt lediglich eine Heuristik dar. Die Wahl von Pivotspalte und Pivotzeile kann auf beliebige Weise geschehen, sofern die Pivotspalte „positive reduzierte Kosten" (S.11f [5]) hat und die Wahl der Pivotzeile zu einer zulässigen Basislösung führt. Es gibt eine Reihe verschiedener Möglichkeiten bei der Auswahl des Pivotelements. Dabei kann nicht nur Einfluss auf die numerische Stabilität genommen werden, sondern auch auf die Rechenzeit und den Speicherbedarf der Implementierung.

Ein LP kann beliebig viele Variablen haben. Der Simplexalgorithmus arbeitet oft aber nur mit einem sehr kleinen Teil dieser Variablen, den Basisvariablen. Nur bei der Auswahl der Pivotspalte werden auch die Nichtbasisvariablen betrachtet. Je nach Auswahlstrategie muss dabei aber nur ein Teil aller Nichtbasisvariablen betrachtet werden. Dieser Umstand wird vom sogenannten revidierten Simplex-Verfahren ausgenutzt. Es speichert lediglich die aktuelle Basismatrix A oder das Inverse sowie ein paar Zusatzinformationen, mit Hilfe derer das Inverse oder die Basismatrix bestimmt werden kann.

Im Internet lassen sich einige Implementierungen des Simplexalgorithmus finden. So wird in [7] eine webbasiertes Werkzeug zur Verfügung gestellt. Dort ist es möglich ein LP direkt im Browser zu optimieren. Als Ergebnis wird zusätzlich zur optimalen Lösung auch das Simplextableau zu jedem Schritt ausgegeben. Allerdings sind keine Gleichungen unter den Nebenbedingungen zugelassen.

Eine Implementierung in Matlab bietet [8]. Allerdings wird hier lediglich das Minimierungsproblem betrachtet. Bessere Funktionalität bietet hier die Matlab-Funktion „linprog“, die allgemein zum Lösen von linearen Programmen verwendet werden kann. Außerdem ist es mit linprog möglich, auch andere Algorithmen wie das Innere-Punkte-Verfahren zu nutzen. (vgl. [4], [6], [1], [5])

7.7 Zusammenfassung

Das Ziel dieser Ausarbeitung war eine Einführung in die Problematik der linearen Optimierung zu geben. Dafür wurde das lineare Optimierungsproblem eingeführt und geometrisch gedeutet. Dabei wurde der Beweis gezeigt, dass sich die optimale Lösung eines linearen Optimierungsproblems in den Eckpunkten des geometrisch beschriebenen Polyeders befindet. Ausgehend von dieser Tatsache wurde die Eckpunkt-Berechnungsmethode vorgestellt. Da sich die Eckpunkt-Berechnungsmethode aufgrund zu hoher Laufzeit nicht für die praktische Anwendung eignet, wurde schließlich der Simplexalgorithmus eingeführt. Der Simplexalgorithmus war zunächst einigen Einschränkungen unterworfen, die sich aber auf später gezeigte Art und Weise einfach auflösen ließen. Schlussendlich wurde noch das duale Programm eines LPs eingeführt und die Verknüpfung zwischen primalem LP und seinem Dual aufgezeigt.

Literaturverzeichnis

1. linprog. Webseite. URL `http://www.mathworks.de/de/help/optim/ug/linprog.html`. Letzter Zugriff 29.05.14
2. Gritzmann, P.: Grundlagen der Mathematischen Optimierung. Springer Fachmedien Wiesbaden (2013)
3. Jarre, F., Stoer, J.: Optimierung. Springer-Verlag Berlin Heidelberg (2004)
4. Korte, B., Vygen, J.: Kombinatorische Optimierung: Theorie und Algorithmen, 2 edn. Springer-Verlag Berlin Heidelberg (2012)
5. Lüdecke, M.: Packungsprobleme unter Toleranzbedingungen. Herbert Utz Verlag (1999)
6. Matousek, J., Gärtner, B.: Understanding and Using Linear Programming. Springer-Verlag Berlin Heidelberg (2007)
7. Priebe, M.: Der Simplex-Rechner. Webseite. URL `http://simplexrechner.matthias-priebe.de/start.php?template=projekte/simplex1.php`. Letzter Zugriff 29.05.2014
8. Sauter, M.: Matlab Programme. Webseite. URL `http://www.math.kit.edu/ianm3/edu/optim12007s/page/programme/`. Letzter Zugriff 29.05.14
9. Schick, K.: Lineares Optimieren. Verlag Moritz Diesterweg/Otto Salle Verlag (1972)

Kapitel 8
Heuristik

Hendrik Grunau

Zusammenfassung Optimierungsprobleme sind im Rahmen dieser Projektgruppe für die Eigenbedarfsoptimierung von zentraler Bedeutung. In dieser Arbeit werden deshalb kombinatorische Optimierungsprobleme als Grundlage für Heuristiken vorgestellt. Anschließend wird auf Heuristische Algorithmen eingegangen, um abschließend Metaheuristiken vorzustellen.

8.1 Einleitung

Für die Eigenverbrauchsoptimierung eines großen Gebäudes sind Optimierungsprobleme von zentraler Bedeutung. In den Gebäuden stehen Energieanlagen und Verbraucher zur Verfügung, welche aufeinander abgestimmt geregelt werden müssen. Als Beispielhaftes Problem könnte eine maximale Energiemenge E_{max} gegeben sein, welche möglichst vollständig verbraucht werden soll. Da jeder Verbraucher x_i einen spezifischen Verbrauch besitzt, stellt sich die Frage welche Kombination von Einheiten einen optimalen Verbrauch erzeugt.

Für die Lösung eines solchen Problems können Heuristiken eingesetzt werden. Dies sind auf ein Problem spezialisierte Suchalgorithmen, welche die beste Variablenkonfiguration zur Problemlösung finden. Dabei ist erstmal nicht relevant, ob diskrete oder reelle Werte verwendet werden.

Im Verlauf dieser Arbeit werden deshalb zunächst kombinatorische Optimierungsprobleme und grundlegende Lösungsmethoden zu diesen vorgestellt. Anschließend werden Metaheuristiken eingeführt und für einen Überblick zunächst Klassifiziert, bevor die Tabu-Suche und genetische Algorithmen als Beispiele für Metaheuristiken genauer betrachtet werden.

Carl von Ossietzky Universität Oldenburg
E-mail: hendrik.grunau@uni-oldenburg.de

8.2 Kominatorische Optimierungsprobleme

Kombinatorische Optimierungsprobleme sind eine Unterklasse der Optimierungsprobleme mit diskreten Variablenräumen. Als Lösung dieser Probleme wird ein Objekt aus einem endlichen oder abzählbar unendlichen Raum gesucht. Dies ist typischerweise eine ganze Zahl, eine Teilmenge, eine Permutation oder eine Graphstruktur.

Definition 8.1 (Kombinatorisches Optimierungsproblem). Ein kombinatorisches Optimierungsproblem $P = (S, f)$ ist definiert durch:

- Eine Menge von Variablen $X = \{x_1, ..., x_n\}$
- Definitionsbereiche $D_1, ..., D_n$
- Nebenbedingungen
- Zu minimierende Zielfunktion(en) $f : D_1 \times ... \times D_n \rightarrow \mathbb{R}^+$
- $S = \{s = \{(x_1, v_1), ..., (x_n, v_n)\} | v_i \in D_i, s \text{ erfüllt alle Nebenbedingungen}\}$

S heißt dabei normalerweise „Such-“oder „Lösungsraum“, und ist die Menge aller möglichen Belegungen oder auch „Kandidatenlösungen“s. Die Qualität eines Kandidats wird über die Fitnessfunktion f bewertet. Je kleiner der Wert der Fitnessfunktion, desto besser ist die jeweilige Lösung. Zur Lösung eines kombinatorischen Optimierungsproblems muss der Kandidat $s^* \in S$ mit minimaler Fitnessfunktion gefunden werden, sodass $f(s^*) \leq f(s) \forall s \in S$ gilt. s^* wird global optimale Lösung von S, f genannt. Dementsprechend ist die Menge $S^* \subseteq S$ die Menge global optimaler Lösungen.[1]

Im Rahmen der Projektgruppe ist eine Menge an Energieverbrauchern in einem Gebäude definiert. Diese sollen nun so kombiniert werden, dass die durch die im Gebäude enthaltenen Generatoren erzeugte Energiemenge optimal ausgenutzt wird. Dadurch ist ein kombinatorisches Optimierungsproblem gegeben.

8.2.1 Das Travelling Salesman Problem

Beispiel eines kombinatorischen Optimierungsproblems ist das NP-schwere „Travelling Salesman Problem (TSP)“.
Das Ziel des Problems ist die kürzeste Reise durch verschiedene Städte zu finden, wobei jede genau einmal besucht werden soll. Als Graph modelliert werden die Städte durch Knoten dargestellt, welche über gewichtete Kanten verbunden sind. Das Gewicht der Kanten kann als Entfernung oder Reisezeit zwischen zwei Knoten gesehen werden. Ein Beispiel ist in Abbildung 8.1 gezeigt.

Für das Problem sind verschiedene Varianten möglich. Normalerweise wird beim TSP von einer Rundreise, Start- und Zielknoten sind identisch, gesprochen, dies muss aber nicht immer der Fall sein. Zusätzlich können die Kanten gerichtet oder ungerichtet sein.[3]

Formalisiert man das Problem aus Abbildung 8.1 nach Definition 8.1 muss zuerst die Variante bestimmt werden. Hier wird abweichend von der klassischen Definition

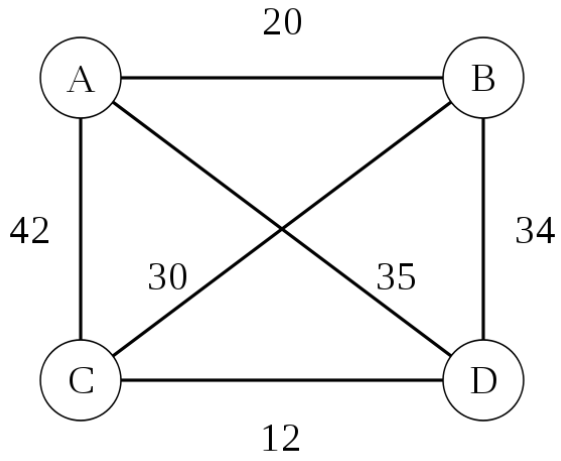


Abb. 8.1 Beispiel eines TSP-Graphen.[5]

gewählt, dass der Startknoten zum Zielknoten verschieden ist und der Graph über ungerichtete Kanten verbunden ist.

Danach können die Variablen X definiert werden. Hierzu wird festgelegt, dass $x_{i,j} \in \{0.1\}$ die Kantenwahl zwischen den Knoten i und j darstellt. Gilt $x_{i,j} = 1$, so ist diese Kante gewählt, andernfalls soll sie unbenutzt sein. Daraus ergeben sich einige Nebenbedingungen:

1. $\sum_{i \in I} x_{i,j} = 1$
2. $\sum_{j \in J} x_{i,j} = 1$
3. $\sum(\sum_{j \in J} x_{i,j}) = N - 1$,

wobei I die Anzahl der eingehenden Kanten vom Knoten, J die Anzahl der ausgehenden Kanten vom Knoten und N die Anzahl der Knoten ist. Die ersten beiden Bedingungen definieren, dass jeder Knoten nur eine eingehende und eine ausgehende Kante besitzt. Die dritte Bedingung legt fest, dass jeder Knoten im Pfad enthalten ist und ein Kantenzug entsteht, da nur von dem Endknoten keine ausgehende Kante mehr gewählt werden soll.

Abbildung 8.2 zeigt einen beispielhaften Pfad, welchen man zu einem Kandidaten s formalisieren kann, welcher alle Nebenbedingungen erfüllt.

$$s = \{(x_{AB}, 1), (x_{AC}, 0), (x_{AD}, 0), (x_{BA}, 0), (x_{BC}, 1), (x_{BD}, 0),$$
$$(x_{CA}, 0), (x_{CB}, 0), (x_{CD}, 1), (x_{DA}, 0), (x_{DB}, 0), (x_{DC}, 0)\}$$

Die eigentlichen Gewichte der Kanten werden nun in der Zielfunktion f als Kosten eingefügt, wodurch sich die Fitness eines Kandidaten durch Addition der Kosten berechnen lässt.

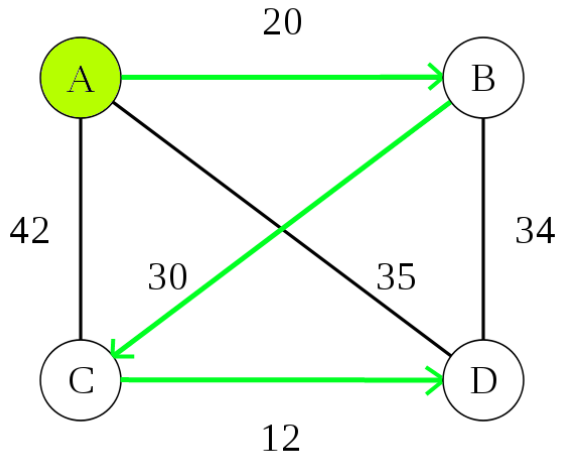


Abb. 8.2 Beispielpfad zu Abbildung 8.1.

$$
\begin{aligned}
f(s) = 1 * 20 + 0 * 42 + 0 * 35 + 0 * 20 + 1 * 30 + 0 * 34 \\
+0 * 42 + 0 * 30 + 1 * 12 + 0 * 35 + 0 * 34 + 0 * 12 \\
= \underbrace{20}_{x_{AB}} + \underbrace{30}_{x_{BC}} + \underbrace{12}_{x_{CD}} \\
= 62
\end{aligned}
$$

Um das Problem zu lösen muss nun der Kandidat s^* mit minimaler Fitnessfunktion f gefunden werden.

8.2.2 Relevanz des TSP

Viele technische „Verfahrprobleme“wie zum Beispiel die Reihenfolge der Bohrungen bei der automatisierten Fertigung einer Leiterplatine lassen sich auf das TSP zurückführen. Betrachtet man nun mögliche Zielfunktionen wird deutlich, das diese speziell auf den Anwendungsfall zugeschnitten sein müssen. So könnte im Fall der Bohrungsablaufsteuerung der Roboter mit zwei Motoren für die Bewegung in der Ebene ausgestattet sein, welche mit einer unterschiedlichen Geschwindigkeit operieren.[3]

Auch das in Abschnitt 8.1 beschriebene Problem lässt sich konzeptuell auf ein TSP zurückführen. So kann man jedem Verbraucher einen Knoten K_i zuweisen, welche über jeweils zwei Kanten mit K_{i-1} und K_{i+1} Verbunden sind. An den beiden „Enden“ werden Start- und Endknoten eingefügt, welche über eine mit 0 gewichtete Kante mit den Verbraucherknoten verbunden sind. Alle Kanten im Graphen sind

in positiv gerichtet, sodass eine lineare Ordnung $Start \rightarrow K_0 \rightarrow ... \rightarrow K_n \rightarrow End$ entsteht. Die beiden Kanten zu jedem K_i sind mit 0 und dem jeweiligen Verbrauch u_i gewichtet. Die Zielfunktion hierzu könnte von der Form $f = E_{max} - cost(\text{Pfad})$ sein, wobei die Kostenfunktion als

$$cost(\text{Pfad}) = \begin{cases} \sum \text{Kanten} & , E_{max} >= \sum \text{Kanten} \\ 0 & , E_{max} < \sum \text{Kanten} \end{cases}$$

definiert ist.

In dieser Form des TSP ist der Weg quasi vorgegeben, da die Reihenfolge der Städte fest gewählt ist. Allerdings wird zu jeder Stadt eine von zwei Abzweigungen gewählt und über die Kostenfunktion bewertet, um den „kürzesten“ Pfad zu finden.

8.3 Algorithmen

Kombinatorische Optimierungsprobleme sind von großer praktischer Wichtigkeit. Daher wurden viele Lösungsalgorithmen entwickelt, welche sich als exakt oder approximativ klassifizieren lassen.
Exakte Algorithmen garantieren durch Überdeckung des kompletten Suchraums die optimale Lösung eines Problems in begrenzter Zeit, sofern die Probleminstanz eine endliche Größe besitzt. Viele kombinatorische Optimierungsprobleme fallen allerdings in die Klasse der NP-schweren Probleme. Für diese existieren keine zeitlich polynomiell von der Probleminstanzgröße abhängigen Lösungsalgorithmen, ausgehend von der Beziehung $P \neq NP$ der Komplexitätsklassen. Für Probleme dieser Komplexität benötigen exakte Methoden im schlechtesten Fall exponentiell von der Instanzgöße abhängige Zeit, was für praktische Anwendungen oft zu lang ist.

Approximative Algorithmen bieten keine Garantie einer optimalen Lösung. Sie zielen darauf ab gute Lösungen in wesentlich kürzerer Zeit als exakte Algorithmen zu berechnen. Dabei lassen sich zwei Grundstrategien, lokale Suche und konstruktive Methoden, erkennen.[1]

8.3.1 Konstruktive Methoden

Das Prinzip der konstruktiven Methoden ist zu einer anfangs leeren Teillösung neue Komponenten hinzuzufügen, bis eine Lösung vollständig ist. Bezogen auf das beispielhafte TSP aus Abbildung 8.1 wird ein konstruktiver Algorithmus zunächst einen Startpunkt wählen, und anschließend immer den kürzeste Weg zum nächsten, unbesuchten Punkt.

Zuerst wird also Punkt A gewählt, welcher zum Pfad $ABCD$ (siehe Abbildung 8.2) mit der Länge 62 vervollständigt wird. Danach folgen, mit der Wahl der neuen

Startpunkte, $|BADC| = 67$, $|CDBA| = 66$ und $|DCBA| = 62$.
Konstruktive Algorithmen sind typischerweise die schnellsten approximativen Methoden, ihre Lösungen im Vergleich allerdings oft schlechter als lokale Suchen[1].

8.3.2 Lokale Suche

Eine lokale Suche startet anders als konstruktive Methoden mit einer Initiallösung. Iterativ wird nun die aktuelle Lösung mit einer besseren aus ihrer (angemessen definierten) Nachbarschaft (siehe Definition 8.2) ersetzt. Ein Nachbar ist anschaulich eine Lösung, welche sich von der aktuellen Lösung nur „wenig“ unterscheidet. Für das TSP könnte die Nachbarschaft zum Beispiel so definiert sein, dass ein Nachbar sich in genau zwei Kanten von dem Referenzpfad unterscheidet. Sei nun für das TSP aus Abbildung 8.1 $|ACBD| = 106$ die aktuelle Lösung. Als Nachbarn dazu findet man unter anderem $|ABCD| = 62$ und $|ADCB| = 77$, woraufhin der beste Nachbar $ABCD$ als neue Lösung gewählt wird, da diese besser ist als die Referenzlösung $(ACBD)$.

Das Suchverfahren hat das Ziel lokale Minima (siehe Definition 8.3) zu finden, wobei ein Abbruchkriterium nicht fest vorgegeben ist. Häufig wird nach einer vorgegebenen Rechenzeit abgebrochen.

Definition 8.2 (Nachbarschaftsstruktur). Eine Nachbarschaftsstruktur ist eine Funktion $N : S \to 2^S$, welche jedem $s \in S$ eine Menge an Nachbarn $N(s) \subseteq S$ zuweist. $N(s)$ wird Nachbarschaft von s genannt.[1]

Definition 8.3 (Lokales Minimum). Ein lokales Minimum hinsichtlich einer Nachbarschaftsstruktur N ist eine Lösung $\hat{s}$ sodass $\forall s \in N(\hat{s}) : f(\hat{s}) \leq f(s)$.
Wenn $f(\hat{s}) < f(s)$ für alle s aus $N(\hat{s})$ gilt wird $\hat{s}$ echtes lokales Minimum genannt.[1]

8.4 Metaheuristiken

In den letzten 20 Jahren wurde eine neue Art approximativer, meist nichtdeterministischer, Algorithmen entwickelt. Diese setzen grundsätzliche heuristische Methoden als Oberstrategie für die Problemlösung ein, um den Suchraum effizient und effektiv zu erforschen. Die dabei benutzten Techniken reichen von einfachen lokalen Suchprozeduren bis zu komplexen Lernprozessen. Diese können Mechanismen enthalten, die das Feststecken in beschränkten Gebieten des Suchraums verhindern.

Kernprinzip der Metaheuristiken ist die Abstraktion des Problems, wodurch sie auf Problemstellungen angepasst werden können.[1]

8.4.1 Klassifizierung

Metaheuristiken können auf verschiedene Arten klassifiziert und beschrieben werden. Je nach Unterscheidungsmerkmal sind verschiedene Klassifizierungen denkbar, wobei jede das Ergebnis eines bestimmten Blickwinkels ist.

Im Folgenden wird ein kurzer Überblick über die wichtigsten Klassifizierungen von Metaheuristiken gegeben, da anhand dieser die Fähigkeiten und Eigenschaften verschiedener Metaheuristiken gut erfasst werden können.

Nature-inspired vs. non-nature inspired

Dies ist die vermutlich intuitivste Art der Klassifizierung von Metaheuristiken. Sie basiert auf dem Ursprung des Algorithmus. In dieser Einordnung existieren Natur inspirierte Algorithmen wie zum Beispiel genetische oder Ameisen Algorithmen, welche den nicht Natur inspirierten Methoden wie einer lokalen Suche gegenüberstehen.

Wenn auch intuitiv hat diese Art der Klassifizierung nur geringe Bedeutung. Dies liegt zum einen daran, dass viele aktuelle, hybride Algorithmen nicht klar einzuordnen sind, da sie gewissermaßen in beide Kategorien gleichzeitig passen. Zum anderen ist es schwierig ein Attribut klar zuzuweisen.[1]

Als Beispiel sei hier die Speichernutzung der Tabu-Suche (siehe Abschnitt 8.5.1) gegeben, welche als Natur inspiriert gesehen werden kann.

Dynamische vs. statische Zielfunktion

Eine weitere Art der Klassifizierung kann anhand der Art, wie die Algorithmen mit der Zielfunktion verfahren stattfinden. Während einige Algorithmen die Zielfunktion unverändert aus der Problemdefinition übernehmen und statisch behandeln, können andere Algorithmen wie zum Beispiel *Guided Local Search (GLS)* diese während der Suche modifizieren.

Die Idee dieses Ansatzes ist aus lokalen Minima zu entkommen indem die Suchlandschaft angepasst wird. Demzufolge wird die Zielfunktion während der Suche verändert, um die durch den Suchprozess erhaltenen Informationen einzubinden.[1]

Eine vs. multiple Nachbarschaftsstrukturen

Metaheuristiken können ebenfalls anhand ihrer Nachbarschaftsstruktur(en) (siehe Definition 8.2) unterschieden werden.

Die meisten Metaheuristiken arbeiten auf einer einzigen Nachbarschaftsstruktur, sodass die Fitnesslandschaft, also die Fitness in Abhängigkeit von der Variablenbelegung, stets die Gleiche ist. Dem gegenüber stehen Metaheuristiken wie die *Variable Neighbourhood Search (VNS)*, welche während der Suche die jeweils aktuelle Nach-

barschaftsstruktur aus einer Menge an Nachbarschaftsstrukturen wählen, und somit die Fitnesslandschaft während der Suche austauschen können.[1]

Gedächtnisnutzende vs. gedächtnislose Methoden

Ein weiteres wichtiges Klassifizierungskriterium für Metaheuristiken ist ihre Benutzung der Suchhistorie, also ob sie ein Gedächtnis benutzen oder nicht. Hierbei ist ein anpassbares Gedächtnis gemeint, im Gegensatz zu einem zum Beispiel bei *Branch & Bound* benutztem starren Gedächtnis.

Gedächtnislose führen einen Markov-Prozess (siehe Definition 8.4), da die nächste Aktion ausschließlich vom aktuellen Zustand des Suchprozesses abhängig ist.

Für die Gedächtnisnutzung existieren mehrere Ansätze. Grundsätzlich kann man zwischen Kurzzeit- und Langzeitgedächtnis unterscheiden. Das Kurzzeitgedächtnis enthält normalerweise kürzlich ausgeführte Entscheidungen. Darunter fallen zum Beispiel ausgewählte Lösungen. Im Langzeitgedächtnis dagegen ist meist eine Sammlung von synthetischen Parametern über die Suche.

Gedächtnisnutzung wird heutzutage als einer der Grundbausteine einer mächtigen Metaheuristik gesehen.[1]

Definition 8.4 (Markov-Prozess). Ein Markov-Prozess ist ein stochastischer Prozess $(X_t)_{0\leq t\leq\infty}$ mit abzählbarem Zustandsraum E, also mit abzählbar vielen möglichen Werten der X_t, bei dem für alle n und alle $t > s > s_n > .. > s_0$ bez. der bedingten Wahrscheinlichkeiten gilt:

$$\begin{aligned} P\{X_t = j | X_s = i, X_{s_n} = i_n, ..., X_{s_0} = i_0\} \\ = P\{X_t = j | X_s = i\} \end{aligned}$$

mit $j, i, i_0, ...i_n \in E$. Die bedingte Wahrscheinlichkeit $P\{X_t = j|X_s = i\}$ heißt Übergangswahrscheinlichkeit (Wahrscheinlichkeit, dass der Prozess zum Zeitpunkt t den Zustand j hat, wenn er zum Zeitpunkt s den Zustand i einnahm).

Ist der Zustandsraum endlich, so wird der Markov-Prozess *endlich* genannt.
Die „Markov-Eigenschaft“ eines stochastischen Prozesses beschreibt, dass die Wahrscheinlichkeit des Übergangs von einem Zustand in den nächstfolgenden von der weiteren Vorgeschichte nicht abhängt.[4]

Population-based vs. single point search

Ausschlaggebend für diese Art der Klassifizierung ist die Anzahl der gleichzeitig benutzten Lösungen einer Metaheuristik.

Algorithmen der Kategorie *single point search* arbeiten zu einem Zeitpunkt t mit nur einer einzigen Lösung, basierend auf dem Prinzip der lokalen Suche. Sie werden auch *trajectory methods* genannt, da sie eine Trajektorie im Suchraum beschreiben.

Unter *population-based* Algorithmen versteht man diejenigen Algorithmen, welche zu einem Zeitpunkt t mit einer Menge an Lösungen arbeiten.

Dies stellt die eindeutigste Klassifizierung von Metaheuristiken dar, wobei aktuell die Hybridisierung dieser Klassen, also die Integration von *single point search* Algorithmen in Populations-basierierte Algorithmen, verfolgt wird. Daher werden die einzelnen Klassen in den Kapiteln 8.5 und 8.6 näher beschrieben.[1]

8.5 Trajektorienmethoden

Diese Art der Metaheuristiken basiert auf dem Prinzip der lokalen Suche. Das bedeutet, dass ein Algorithmus dieser Kategorie mit einer Initiallösung startet, um die optimale Lösung zu finden. In jedem Schritt durch eine bessere ersetzt wird, wie in Abschnitt 8.3.2 beschrieben. Anhand dieser kann man eine Trajektorie im Suchraum beschreiben.[1] Anschaulich kann man dazu die Fitnessfunktion eines Problems grafisch (siehe Abbildung 8.3) darstellen, und einen Vektor von Initiallösung zu gefundenen Optimum einzeichnen.

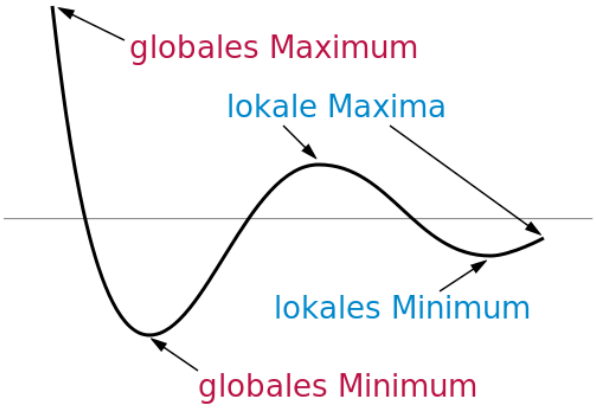


Abb. 8.3 Beispiel eines Suchraums mit verschiedenen Extrema. [2]

Ein Problem dieser Methoden ist, dass die Güte der gefundenen Lösung stark von der Initiallösung abhängig ist. Anhand des Beispiels aus Abbildung 8.3 ist leicht zu erkennen, dass eine Initiallösung rechts des lokalen Maximums das lokale Minimum als optimale Lösung erhält. Wählt man dagegen eine Initiallöung links des lokalen Maximums, so wird das globale Minimum als optimale Lösung gefunden.

Um dieses Problem zu überwinden müsste ein Algorithmus zunächst das lokale Maximum als Lösung akzeptieren, welches allerdings eine schlechtere Fitness als die Initallösung hat. Der in Abschnitt 8.5.1 vorgestellte Algorithmus zeigt, wie dieses Problem durch Gedächtnisnutzung gelöst werden kann.

8.5.1 Tabu-Suche

Die Tabu-Suche stellt eine Erweiterung der grundsätzlichen lokalen Suche dar. Sie ermöglicht lokale Optima wieder zu verlassen, indem für kurze Zeit die Verschlechterung der Fitness zugelassen wird. Um anschließend nicht wieder in das eben verlassene lokale Minimum zurückzufallen wird eine Tabu-Liste geführt, in welcher bereits besuchte Lösungen für eine Zeit gespeichert, und bei der Lösungssuche ausgelassen werden.[6]

Der Algorithmus

Der Erste Schritt einer Tabu-Suche ist die Initiallösung als aktuelle Lösung x zu setzen. Diese wird dann als beste Lösung x_{best} gespeichert und in die Tabu-Liste mit der Wartezeit t_w eingetragen. Anschließend wird wie folgt iteriert:

Zuerst werden alle Nachbarn $N(x)$ der aktuellen Lösung gesammelt. Alle Lösungen der Tabu-Liste werden nun aus $N(x)$ entfernt. Aus den verbliebenen Lösungen der Nachbarschaftsmenge wird nun diejenige Lösung x' mit der besten Fitness ausgewählt und die Wartezeit aller Lösungen in der Tabu-Liste um eins reduziert, wobei die Lösungen mit $t_w = 0$ aus der Liste entfernt werden. Hiernach kann die gewählte Lösung x' in die Tabu-Liste eingetragen werden. Gilt $f(x') < f(x_{best})$ wird mit $x_{best} = x'$ die beste Lösung aktualisiert. Abschließend wird die aktuelle Lösung für die nächste Iteration mit $x = x'$ belegt.

Wie bei der grundsätzlichen lokalen Suche ist ein Abbruchkriterium nicht fest vorgegeben. Dies trifft auch auf die Wartezeit t_w zu. Diese muss so klein gewählt sein, dass alte Lösungen rechtzeitig wieder freigegeben werden, darf allerdings auch nicht zu klein sein, da sonst die gleichen Lösungen zyklisch immer wieder geprüft werden könnten.[6]

8.6 Populationsbasierte Algorithmen

Populationsbasierte Algorithmen arbeiten in jeder Iteration mit einer Menge an Lösungen, einer Population von Individuen. Dabei bieten sie eine natürliche, eigenleitende Art den Suchraum zu erforschen. Die Leistungsfähigkeit hängt dabei allerdings stark von der Populationsmanipulation ab.

8.6.1 Evolutionäre Algorithmen

Evolutionäre Algorithmen gehören zu den am häufigsten studierten populationsbasierten Algorithmen. Ihre Lösungsstrategie basiert auf den Prinzipien der natürli-

chen Evolution. Die Codierung einer Lösung wird dementsprechend als „Chromosom“ bezeichnet.

Startpunkt für den Optimierungsprozess ist dabei eine Menge von Initiallösungen, die Startpopulation. In jeder Iteration werden aus sogenannten „Eltern“ neue Individuen, „Kinder“, durch Rekombination erzeugt. Diese mutieren, bevor anhand der Fitness eine neue Elterngeneration selektiert wird. Als Abbruchkriterium kann eine maximale Anzahl an Generationen oder optimale Fitness gewählt werden.

Zunächst sei die Bitstringoptimierung als Beispielproblem gegeben. Dabei soll die Anzahl der 1 auf einem Bitstring der Länge l maximiert werde, die Anzahl der 0 minimiert. Für die Erläuterung der vier grundlegenden Mechanismen genügt hier die Länge 3.

Rekombination

Bei der Rekombination werden Paare von Individuen der Elterngeneration ausgewählt. Ihre Chromosomen werden dann an $1 \leq n$ gleichen, zufällig gewählten Stellen geteilt und anschließend neu kombiniert. Dabei entstehen zwei neue Individuen.
Als Beispiel seien die beiden Eltern 100 und 001 gegeben. Diese werden nun an einer Stelle rekombiniert:

$$100, 001 \xrightarrow[recomb.]{} 101, 000$$

Mutation

In jeder Iteration werden aus Eltern neue Kinder erzeugt, welche mutieren sollen. Der Mutationsoperator ändert das Chromosom des gerade ausgewählten Individuums zufällig an $1 \leq m$ Stellen. Die Änderung kann dabei ein zufälliger Wert, das Inverse des Chromosoms, das Hinzufügen eines Gaussverteilten Zufallswertes oder ähnliches sein.

Hier wird eine Bitflip-Mutation gewählt, angewendet auf eines der gerade erzeugten Kinder:

$$101 \xrightarrow[mutate]{} 111$$

Fitnessevaluation und Selektion

Wie bereits erwähnt stellt jedes Individuum eine Lösung des Optimierungsproblems dar. Dadurch muss für jedes Individuum die Fitnessfunktion ausgewertet werden.

Der Selektionsprozess für die nächste Generation von Individuen benötigt eine Auswahl Individuen als neue Eltern aus der Population. Dies geschieht anhand der Fitness der Individuen. Dabei existieren verschiedene Strategien. So können die aktuellen Eltern in diesen Prozess eingebunden sein, oder nicht. Zusätzlich müssen nicht

nur die besten Individuen ausgewählt werden. So können Individuen mit schlechterer Fitness dafür sorgen, dass die Suche nicht vorzeitig in lokalen Minima endet.

8.7 Fazit

Als approximative Algorithmen sind Metaheuristiken in der Lage schnell relativ gute Lösungen für Optimierungsprobleme zu finden, wodurch sie sich für den Einsatz in Echtzeitanwendungen eignen. Ihre Qualität und Geschwindkeit hängt dabei allerdings stark von der jeweiligen Implementierung und Problemdefinition ab.

Durch die Abstraktion von der Probleminstanz können Metaheuristiken auf ein Optimierungsproblem angepasst werden, was allerdings nicht heißt das dies für alle Metaheuristiken mit gleichem Aufwand geschehen kann. Die Tabu-Suche (siehe Abschnitt 8.5.1) kann zum Beispiel einfacher auf das TSP (siehe Abschnitt 8.2.1) angepasst werden als ein genetischer Algorithmus (siehe Abschnitt 8.6.1).

Im Rahmen der Projektarbeit muss daher zunächst eine ausreichende Definition des Optimierungsproblems der Verbraucherkombination gefunden werden. Anschließend können anhand der Klassifizierungen für Metaheuristiken Merkmale identifiziert werden, durch welche das Entkommen aus lokalen Minima gesichert werden soll. Abschließend kann eine für die Problemrepräsentation geeignete Metaheuristik gewählt, angepasst und implementiert werden, um die Steuerung der Verbraucher zu realisieren.

Literaturverzeichnis

1. Blum, C., Roli, A.: Metaheuristics in Combinatorial Oprimization: Overview and Conceptual Comparison. ACM Computing Surveys **35** (2003)
2. Georg-Johann: Extrema_example_de. Webseite. URL `http://commons.wikimedia.org/wiki/File:Extrema_example_de.svg`. Letzter Zugriff 30.05.2014
3. Grötschel, M.: Schnelle Rundreisen: Das Travelling Salesman-Problem. ZIB-Report **57** (2005)
4. Kramps, U.: Gabler Wirtschaftslexikon, Stichwort: Markov-Prozess. Webseite. URL `http://wirtschaftslexikon.gabler.de/Archiv/11201/markov-prozess-v10.html`. Letzter Zugriff 30.05.2014
5. Sdo: Weighted_K4. Webseite. URL `http://en.wikipedia.org/wiki/File:Weighted_K4.svg`. Letzter Zugriff 16.05.2014
6. Sonnenschein, M., Vogel, U.: Skript zur Vorlesung Algorithmen und Datenstrukturen (2011). 6. Auflage

Kapitel 9
Ganzzahlige lineare Optimierung

Sönke Martens

Zusammenfassung Im Rahmen dieser Seminararbeit wird eine Einführung in das Thema der ganzzahligen linearen Programmierung geboten. Dazu werden die Unterschiede von linearer Programmierung und ganzzahliger linearer Programmierung aufgezeigt und motiviert, wieso die Verwendung von ganzen Zahlen bei der Modellierung von Problemen häufig notwendig ist. Aus Beobachtungen über untere und obere Schranken von linearen Problemen werden einfache Verfahren, wie der Branch-and-Bound Algorithmus, hergeleitet. Außerdem wird gezeigt, wie dieser unter Verwendung von speziellen Bedingungen, den Cutting Planes, verbessert werden kann.

9.1 Einleitung

Durch die zunehmende Verbreitung dezentraler Energieerzeugungsanlagen wird das Stromnetz zusätzlicher Belastung ausgesetzt. Denn diese Art von Erzeugung ist kaum steuerbar und somit wird auch dann Strom produziert, wenn er nicht benötigt wird. So führt beispielsweise der Einsatz von PV-Anlagen zu einer erhöhten Erzeugung in den Mittagsstunden. Dies könnte abgemindert werden, wenn durch den Einsatz von Energiespeichern und flexiblen Verbrauchern Erzeugungsspitzen schon vom Betreiber selbst kompensiert werden. Beispielweise indem Geräte, wie Waschmaschinen, zur Mittagszeit automatisch gestartet werden. Durch die Nutzung des selbst erzeugten Stroms werden zusätzlich die Stromkosten für den Betreiber der Anlage gesenkt. Somit kommt es sowohl dem globalen Stromnetz als auch dem Betreiber zu Gute, wenn der erzeugte Strom selbst verbraucht wird. Das Ziel, möglichst viel erzeugten Strom lokal selbst zu verbrauchen, wird Eigenverbrauchsoptimierung genannt. [2]

Carl von Ossietzky Universität Oldenburg
E-mail: soenke.martens@uni-oldenburg.de

Probleme, wie die Eigenverbrauchsoptimierung, lassen sich nicht mittels einfacher linearer Programmierung lösen, weil diese nur reellwertige Variablen kennt. Jedoch sind bei der Modellierung von Geräten mit einer begrenzten Anzahl von Gerätemodi ganze Zahlen notwendig. Wollen wir beispielsweise über eine positive Variable x ausdrücken, dass ein Gerät entweder an (1) oder aus (0) ist, können wir das mit ganzen Zahlen über die Bedingung $x \leq 1$ ausdrücken. Dahingegen werden bei der linearen Programmierung Zahlen zwischen 0 und 1 durch diese Bedingung nicht ausgeschlossen. Deshalb benötigen wir stattdessen Methoden der ganzzahligen linearen Programmierung, um solche Probleme zu lösen.

Im Rahmen der Seminararbeit werden unterschiedliche Lösungsverfahren vorgestellt, die Probleme der ganzzahligen linearen Programmierung lösen können. Zu Beginn wird in Abschnitt 9.2 beschrieben, wie Probleme der linearen und ganzzahligen linearen Programmierung aufgebaut und definiert sind. Darauf aufbauend werden in Abschnitt 9.3 unterschiedliche Verfahren der ganzzahligen linearen Programmierung, wie z.B. Branch-and-Bound, erläutert und mit Hilfe von Beispielen verdeutlicht. Abschließend wird in Abschnitt 9.4 in einem Fazit der Inhalt der Seminararbeit zusammengefasst und ein Ausblick geboten.

9.2 Problembeschreibung

Bei der linearen Programmierung wird versucht eine lineare Zielfunktion zu maximieren. Dazu werden positive Werte für Variablen $x_i, ..., x_n$ gesucht, so dass die Zielfunktion maximal ist. Eine solche Zielfunktion hat die Form (9.1), wobei $c_i, x_i \in \mathbb{R}$. [3] [6]

$$\begin{aligned} \max z := \quad & c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n \\ & x_1, x_2, \ldots, x_n \geq 0 \end{aligned} \tag{9.1}$$

Zum Beispiel können bei einer solchen Zielfunktion die Konstanten c_i den Gewinn beim Verkauf einer Ware und x_i die Stückzahlen dieser Ware angeben. In vielen Fällen reicht eine Zielfunktion zur Beschreibung des Problems nicht aus, sondern es existieren weitere Bedingungen die unsere Variablen x_i erfüllen müssen. So wollen wir zum Beispiel für obiges Beispiel formulieren können, dass die Stückzahlen nach oben begrenzt sind, da der betrachtete Betrieb nur eine begrenzte Menge an Kapital zur Verfügung hat. Um solche Bedingungen zu ermöglichen, wird die Problembeschreibung um lineare Bedingungen mit den Konstanten $A_{ij}, b_i \in \mathbb{R}$ erweitert. (Siehe (9.2))

Zur Lösung von Problemen der linearen Programmierung sind verschiedene Lösungsmethoden, wie z.B. die Innere-Punkt-Methode und das Simplex-Verfahren, vorhanden. Für einige Varianten der Inneren-Punkt-Methode wurde gezeigt, dass sie alle linearen Probleme in polynomieller Zeit lösen können, und die lineare Programmierung somit eine polynomielle Komplexität hat [3]. Im Gegensatz zu der Inneren-

$$
\begin{aligned}
\max z := \quad & c_1 \cdot x_1 + c_2 \cdot x_2 + \cdots + c_n \cdot x_n \\
\text{subject to} \quad & A_{11} \cdot x_1 + A_{12} \cdot x_2 + \cdots + A_{1n} \cdot x_n \leq b_1 \\
& A_{21} \cdot x_1 + A_{22} \cdot x_2 + \cdots + A_{2n} \cdot x_n \leq b_2 \\
& \qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
& A_{n1} \cdot x_1 + A_{n2} \cdot x_2 + \cdots + A_{nn} \cdot x_n \leq b_n \\
& x_1, x_2, \ldots, x_n \geq 0
\end{aligned} \tag{9.2}
$$

Punkt-Methode benötigt das Simplex-Verfahren für manche Probleme exponentielle Zeit. Dennoch wird das Simplex-Verfahren in den meisten Fällen verwendet, da es bei den meisten Problemen zeitlich effizienter ist. [3]

Diese Verfahren lassen sich nicht benutzen, wenn wir das Problem so verschärfen, dass alle oder manche Variablen x_i ganzzahlig sein sollen. In diesem Fall spricht man von ganzzahliger und gemischt-ganzzahliger linearer Programmierung oder im Englischen von *integer linear programming* und *mixed integer linear programming*. Die Verschärfung des Problems mit ganzzahligen Variablen führt dazu, dass sich die Komplexität des Problems auf *NP-schwer* erhöht [7], obwohl dies der Vorstellung widerspricht, dass ganzzahliges Rechnen einfacher ist als das Rechnen mit reellen Zahlen. Leider sind ganze Zahlen häufig notwendig. Beispielsweise wenn wir nur ganze Stückzahlen zulassen wollen, da halbe Waren unverkäuflich sind oder wenn wir Eigenschaften wie verschiedene Modi von Geräten mit einer Variablen modellieren möchten. Es werden also ebenfalls Verfahren benötigt, um Probleme der ganzzahligen linearen Programmierung lösen zu können.

9.3 Lösungsverfahren

Es sind verschiedene Verfahren vorhanden, um ganzzahlige lineare Programmierung zu lösen. Das grundlegende Verhalten der verschiedenen Vorgehensweisen ist dabei oft sehr ähnlich. Die Verfahren ignorieren zunächst die Ganzzahligkeitsbedingung und lösen das vereinfachte Problem mittels eines Verfahrens der linearen Programmierung. Diese Vereinfachung auf die lineare Programmierung wird *Relaxierung* genannt [3]. Nachdem die Relaxierung beispielsweise mit einem Simplex-Solver gelöst wurde, überprüft das Verfahren anschließend, ob die optimale Lösung der Relaxierung ganzzahlig ist. Ist dies der Fall, ist das Problem gelöst und das Verfahren terminiert. Ansonsten wird der Suchraum weiter eingegrenzt und anschließend wieder die Relaxierung gelöst usw.

Beim Branch-and-Bound Verfahren wird das Problem begrenzt, indem der Suchraum in Teilprobleme zerlegt wird. Dahingegen führt die Cutting Plane Methode zusätzliche Bedingungen ein, welche die bisher beste nicht-ganzzahlige Lösung aus dem Suchraum entfernen, dabei aber alle ganzzahligen Lösungen im Suchraum belassen. Die Branch-and-Cut Methode ist eine Kombination des Branch-and-Bound

Verfahrens und des Cutting Plane Algorithmus, welche in vielen Fällen effizienter ist als beide Algorithmen für sich alleine [6]. Diese drei Algorithmen werden in den folgenden Abschnitten detailliert beschrieben und erläutert.

Neben den hier vorgestellten Algorithmen gibt es weitere Methoden, wie beispielsweise die Branch-and-Price Methode. Die Branch-and-Price Methode weicht im Vorgehen von Branch-and-Bound insofern ab, als sie zunächst nicht alle Variablen betrachtet, sondern mit der Zeit Variablen hinzunimmt, um Schritt für Schritt das Problem zu lösen. Dies kann insbesondere dann sinnvoll sein, wenn sehr viele Variablen vorhanden sind [1]. Auch denkbar ist es, dass Problem mit Heuristiken zu lösen, die insbesondere dann sinnvoll sind, wenn die Lösung nicht unbedingt optimal sein muss. Bevor auf die verschiedenen Algorithmen eingegangen wird, werden in Abschnitt 9.3.1 Beobachtungen gemacht, die bei der Lösung und Eingrenzung des Problems helfen.

9.3.1 Beobachtungen

Um ganzzahlige lineare Programmierung zu lösen, kann wie oben beschrieben die Relaxierung auf ein lineares Problem mit reellen Zahlen genutzt werden. Die Lösung der Relaxierung bietet dabei nicht nur einen Kandidaten zum Test auf Ganzzahligkeit, sondern liefert uns Informationen über das ganzzahlige Problem, die wir nutzen können. Nehmen wir an $R \in \mathbb{R}$ sei die Menge aller Wertepaare, welche die Bedingungen der Relaxierung erfüllen und $N \in \mathbb{N}$ die Menge aller Wertepaare, welche die Bedingungen erfüllen und ganzzahlig sind. Da das ganzzahlige Problem eine Verschärfung seiner Relaxierung ist, gilt $N \subseteq R$. Daraus kann man schließen, dass das Optimum der Relaxierung mindestens so gut wie das Optimum des ganzzahligen Problems ist. Folglich ist die Lösung der Relaxierung eine obere Schranke für das ganzzahlige Problem und kann somit als Abschätzung dienen. [5]

Dies wird im Folgenden an einem Beispiel mit zwei Variablen und zwei Bedingungen gezeigt:

$$\begin{aligned} \max z := {} & x_1 + 2 \cdot x_2 \\ \text{subject to} \quad & 4 \cdot x_1 + 3 \cdot x_2 \leq 23 \\ & -2 \cdot x_1 + 3 \cdot x_2 \leq 3 \\ & x_1, x_2 \geq 0, \textit{integer} \end{aligned} \tag{9.3}$$

Der Lösungsraum ist in Abbildung 9.1 gezeigt. Sichtbar sind die beiden Bedingungen, die den Lösungsraum eingrenzen, und die ganzzahligen Lösungen, die diese Bedingungen erfüllen. Die optimale Lösung für die Relaxierung ist der Punkt $(3\frac{1}{3}, 3\frac{2}{9})$ mit einem Wert von $z = 9\frac{7}{9}$. Diesen kann man beispielsweise mit dem Simplex-Verfahren berechnen oder in diesem einfachen Fall auch grafisch ermitteln. Dazu wird die Zielgleichung aufgetragen und solange nach unten verschoben bis sie den Raum, den die Bedingungsgleichungen begrenzen, schneidet. Dies ist genau

der Punkt $(3\frac{1}{3}, 3\frac{2}{9})$. Die Zielfunktion teilt dabei den Lösungsraum in zwei Teilräume. Einen Teilraum mit schlechteren Lösungen und einen Teilraum mit besseren Lösungen. In diesem Fall, sind alle ganzzahligen Lösungen im Bereich mit schlechteren Lösungen. $(3\frac{1}{3}, 3\frac{2}{9})$ ist also eine obere Schranke für das Beispiel.

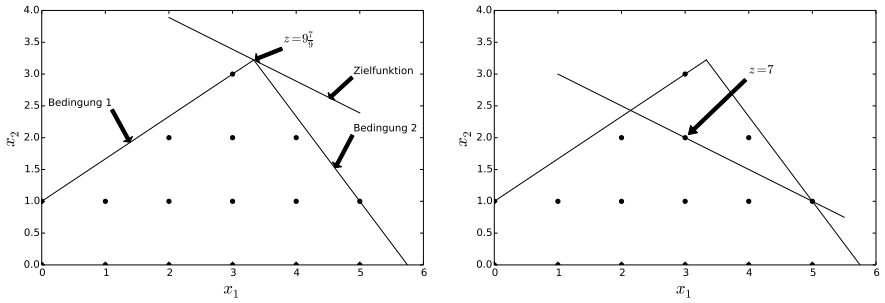


Abb. 9.1 Grafische Darstellung eines ganzzahligen Problems. Der Lösungsraum wird durch die Bedingungen eingegrenzt. (Links) Das Optimum der Relaxierung ist eine Obergrenze für das ganzzahlige Problem. (Rechts) Ganzzahlige Lösungen trennen das Problem in schlechtere und bessere Lösungen.

Wird im Laufe eines Verfahrens eine ganzzahlige Lösung gefunden, welche das Problem löst, teilt diese Lösung den Raum ebenfalls in zwei Teilräume mit besseren und schlechteren Lösungen. Dies ist ebenfalls grafisch sichtbar, indem die Zielfunktion auf die gefundene Lösung aufgetragen wird. Für das obige Beispiel, erfüllt beispielsweise die Lösung $(3, 2)$ das ganzzahlige Problem mit einem Wert von 7. (Siehe Abbildung 9.1 rechts) Wurde dieser Wert im Rahmen eines Verfahrens ermittelt, kann der Algorithmus danach alle Lösungen mit schlechteren Werten als Kandidaten für das Optimum ausschließen. Solch ein Wert stellt also eine untere Schranke dar. Diese untere Schranke lässt sich noch verstärken, indem wir aus der Zielgleichung berechnen, wie groß die Minimalverbesserung zum nächst besseren Wert mindestens ist. [3]

Für das Beispiel ist dies einfach ablesbar. Aus der Zielgleichung kann direkt abgelesen werden, dass die geringste Änderung bei einer Veränderung von x_1 oder x_2 1 ist. Folglich können wir 8 als untere Schranke annehmen, wenn wir zur gefundenen Lösung 7 beim Punkt $(3, 2)$ die geringste Änderung 1 addieren. Die Beobachtungen zu unteren und oberen Schranken werden in den folgenden Algorithmen verwendet, um Teile des Lösungsraum auszuschließen und den Ablauf somit zu beschleunigen.

9.3.2 Branch-and-Bound

Der Branch-and-Bound Algorithmus verfolgt eine divide-and-conquer-Strategie, d.h. ein schwer zu lösendes Problem wird in leichter zu lösende Teilprobleme geteilt. Im Fall des Branch-and-Bound ist das Vorgehen folgendes: Zunächst wird der Algorithmus initialisiert, indem eine untere Schranke $\underline{z} = -\infty$ erzeugt wird. Diese dient später als Abbruchkriterien für Teilprobleme und wird im Laufe des Algorithmus verbessert. Falls gewünscht kann zur Verbesserung der Abschätzung der unteren Grenze eine Minimalverbesserung Δz berechnet werden. (Siehe Abschnitt 9.3.1) Außerdem existiert eine Menge ILP, in der alle bisher nicht betrachteten Teilprobleme gespeichert sind. Diese enthält zunächst nur das Ursprungsproblem. [3]

Nach der Initialisierung startet der Algorithmus eine Schleife, in der die Probleme aus der Menge ILP abgearbeitet werden. Dabei können drei Fälle auftreten: Die Lösung des Teilproblems wird als untere Schranke gespeichert, verworfen oder falls beides nicht möglich ist, in zwei Teilprobleme geteilt. Im Folgenden wird das Vorgehen in der Schleife begründet erläutert. Für einen besseren Überblick ist in Abbildung 9.2 der Algorithmus in Pseudocode skizziert.

```
1  z = −∞
2  ILP = {Anfangsproblem}
3  while ILP ≠ ∅ do
4      Entferne ein Problem P aus ILP
5      Löse Relaxierung von P
6      if P lösbar then
7          z = Lösung der Relaxierung
8          if z ≥ z then
9              if Variablen x_i ganzzahlig then
10                 z = z
11                 Speichere Variablen x_i
12             else
13                 Teile Problem P in P_1 und P_2
14                 ILP = ILP ∪ {P_1, P_2}
```

Abb. 9.2 Branch-and-Bound Algorithmus in Pseudocode (angelehnt an [3] und [6])

Als erstes wird in Zeile 4 des Algorithmus ein Problem P aus der Menge der offenen Probleme ILP entfernt und danach die Relaxierung von P gelöst. Falls keine Lösung existiert, d.h. das Problem P unerfüllbar ist, wird P verworfen und wir beginnen die Schleife von vorne (Siehe Zeile 6). Die Schleife wird ebenfalls von vorne begonnen, wenn das Optimum z der Relaxierung kleiner ist als unsere momentane Untergrenze $\underline{z}$. Denn in diesem Fall ist klar, dass alle möglichen Lösungen in P kleiner sind als der bisher beste gefundene Wert. Wenn jedoch z größer als $\underline{z}$ ist, ist die Lösung der Relaxierung ein geeigneter Kandidat für die Lösung des ganzzahligen Problems. Deshalb werden anschließend in Zeile 9 die Variablen x_i auf Ganzzahligkeit überprüft. Ist dies der Fall, wurde eine neue beste Lösung gefunden und $\underline{z}$ wird

aktualisiert. Ist eine Variable der Lösung nicht ganzzahlig, können wir nicht wissen, ob im Lösungsraum des Teilproblems eine ganzzahlige Lösung existiert, die besser als die bisherige untere Schranke ist. In diesem Fall wird das Teilproblem in zwei Teilprobleme geteilt (Zeile 13). Die Teilprobleme sind mit dem vorherigen Problem bis auf eine zusätzliche Bedingung identisch. Das erste Teilproblem besitzt die zusätzliche Bedingung $x_i \leq k$ mit $k \in \mathbb{N}$ und das zweite Problem eine Bedingung der Form $x_i \geq k + 1$. Diese Bedingungen bewirken, dass die Teilprobleme jeweils einen disjunkten Teil des Lösungsraums begrenzen. Nach der Teilung, werden die beiden Probleme zu ILP hinzugefügt und die Schleife beginnt erneut. Diese läuft solange, bis keine offenen Probleme in der Menge ILP vorhanden sind. Sind keine Probleme mehr offen, terminiert der Algorithmus mit $\underline{z}$ als Optimum für das ganzzahlige Problem.

Im Weiteren wird die Funktionsweise des Algorithmus am Beispiel aus Abschnitt 9.3.1 verdeutlicht. Zunächst wird $\underline{z}$ initialisiert und das Ursprungsproblem (9.3) zur Menge ILP hinzugefügt. Danach beginnt die Schleife und das Problem (9.3) wird entfernt und gelöst. Die Lösung $9\frac{7}{9}$ ist größer als $-\infty$. Jedoch ist der Punkt $(3\frac{1}{3}, 3\frac{2}{9})$ nicht ganzzahlig. Das Problem wird also in zwei Teilprobleme geteilt. Dazu wird beispielsweise x_1 und $k = 3$ gewählt und es entstehen die beiden neuen Probleme (9.4) und (9.5). Beide Probleme sind in Abbildung 9.3 als Formeln und in Abbildung 9.4 grafisch dargestellt.

$$
\begin{aligned}
&\max z := x_1 + 2 \cdot x_2 && (9.4)\\
&\text{subject to} \quad 4 \cdot x_1 + 3 \cdot x_2 \leq 23\\
&\qquad -2 \cdot x_1 + 3 \cdot x_2 \leq 3\\
&\qquad x_1 \leq 3\\
&\qquad x_1, x_2 \geq 0, \textit{integer}
\end{aligned}
\qquad\qquad
\begin{aligned}
&\max z := x_1 + 2 \cdot x_2 && (9.5)\\
&\text{subject to} \quad 4 \cdot x_1 + 3 \cdot x_2 \leq 23\\
&\qquad -2 \cdot x_1 + 3 \cdot x_2 \leq 3\\
&\qquad x_1 \geq 4\\
&\qquad x_1, x_2 \geq 0, \textit{integer}
\end{aligned}
$$

Abb. 9.3 Teilung von Beispiel (9.3) in zwei disjunkte Teilprobleme

Im nächsten Schritt wird Problem (9.4) gewählt und seine Relaxierung mit dem Simplex-Verfahren gelöst. Es ergibt sich für das Teilproblem ein Optimum von $z = 9$ beim Punkt $(3, 3)$. Da Punkt $(3, 3)$ ganzzahlig ist und $9 \geq -\infty$ gilt, wird 9 als neue untere Schranke gespeichert. Anschließend wird das letzte Problem (9.5) aus ILP betrachtet, dessen Optimum mit $8\frac{2}{3}$ jedoch kleiner als die untere Schranke 9 ist und somit verworfen werden kann. Die Schleife terminiert mit dem Optimum 9 beim Punkt $(3, 3)$, weil keine weiteren Probleme in ILP vorhanden sind.

Soll der Branch-and-Bound Algorithmus implementiert werden, sind vorher einige Überlegungen notwendig. Denn beispielsweise haben sowohl die Wahl eines Teilproblems aus der Menge der offenen Probleme als auch die Wahl einer Variable bei der Teilung Einfluss darauf, wie schnell der Algorithmus das Problem optimieren kann [3]. In beiden Fällen kann es Sinn machen, bereits vorhandenes problemspezifisches Wissen zu nutzen, um Teilprobleme und Variablen zu bevorzugen, bei der eine schnellere Terminierung des Problems vermutet wird. Ebenfalls kann es sinn-

voll sein, Teilprobleme nicht in Breitensuche sondern in Tiefensuche zu durchlaufen. Das bedeutet, dass nach der Teilung eines Problems direkt ein erzeugtes Teilproblem gewählt wird, damit schneller ein einfaches Teilproblem eine untere Schranke liefern kann. [3]

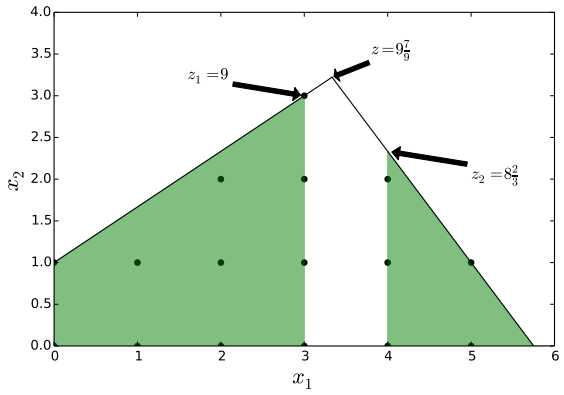


Abb. 9.4 Aufteilung des Problems in zwei Teilprobleme (grün dargestellt) mit dem Branch-and-Bound Verfahren

9.3.3 Cutting Planes

Im Folgenden wird eine besondere Variante von Bedingungen vorstellt, die sogenannten Cutting Planes. Mit diesen lässt sich ein eigenständiger Algorithmus zur Lösung von ganzzahligen linearen Problemen erstellen. Sie können aber auch in anderen Verfahren wie dem Branch-and-Bound Verfahren eingesetzt werden, um diese zu beschleunigen [6]. Cutting Planes haben die folgende Definition:

Definition 1 (Cutting Plane) *Eine Cutting Plane ist eine Bedingung der Form*

$$a_1 \cdot x_1 + a_2 \cdot x_2 + \cdots + a_n \cdot x_n \leq b$$

für die bzgl. eines ganzzahligen linearen Problems gilt:

1. *Das reellwertige Optimum der Relaxierung erfüllt die Bedingung nicht.*
2. *Alle ganzzahligen Lösungen des Problems erfüllen die Bedingung.*

Cutting Planes schränken zwar den Lösungsraum der ganzzahligen Lösungen nicht ein, jedoch entfernen sie reellwertige Lösungen (mindestens das bisherige Optimum) aus dem Lösungsraum der Relaxierung, wenn sie zu den Bedingungen

des Problems ergänzt werden. Einfach ausgedrückt, schneidet die Cutting Plane reellwertige Lösungen ab, so dass im besten Fall mit Hilfe der Cutting Plane ein Problem entsteht, dessen Relaxierung das gleiche Optimum wie das ganzzahlige Problem hat. Dabei unterscheiden sich Cutting Planes in ihrer Güte. Im schlimmsten Fall entfernt die Cutting Plane nur das Optimum der Relaxierung und wenige umliegende Lösungen. Im besten Fall liegen die Cutting Planes auf dem konvexen Polyeder, den die ganzzahligen Lösungen aufspannen. Dann entfernt eine Cutting Plan den größtmöglichen Anteil von Lösungen, die zum ganzzahligen Problem nicht beitragen [6]. In Abbildung 9.5 ist für das zweidimensionale Beispiel das konvexe Vieleck gezeigt, das die ganzzahligen Lösungen umspannt und auf dem die optimalen Cutting Planes liegen.

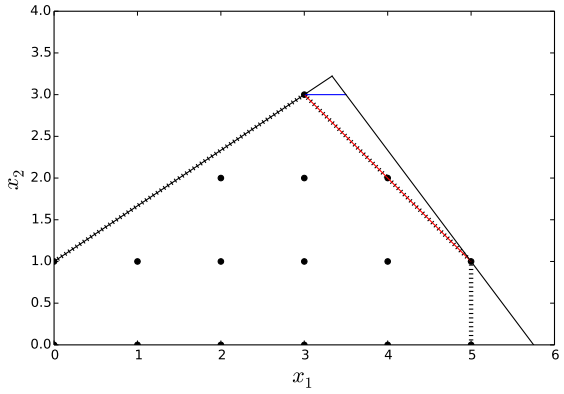


Abb. 9.5 Eine optimale Cutting Plane (rot) liegt auf der gestrichelten Kante des konvexen Polyeders, welches das Problem bildet. Oberhalb der optimalen Cutting Plane liegt eine zweite blaue nicht-optimale Cutting Plane.

Es existieren unterschiedliche Arten von Cutting Planes. Zum einen gibt es problemspezifische Cutting Planes, die aus Informationen des Problems hergeleitet werden. Diese können auch genutzt werden, wenn das vorliegende Problem auf ein anderes Problem übertragen wird, für das Cutting Planes bereits bekannt sind. Beispielsweise sind solche problemspezifischen Bedingungen für das Problem des Handlungsreisenden bereits vorhanden. Da diese Cutting Planes für ein vorhandenes Problem optimiert sind, weisen sie eine hohe Güte aus und sind daher, falls bekannt oder herleitbar, zu bevorzugen. Im Gegensatz zu den problemspezifischen Cutting Planes sind Klassen problemübergreifender Cutting Planes meistens schwächer. Dessen ungeachtet sind diese auch dann einsetzbar, wenn das Finden problemspezifischer Cutting Planes nicht möglich ist. Aus diesem Grund wird im Weiteren eine Variante allgemeingültiger Cutting Planes, die *Gomory Cuts*, vorgestellt. [6]

Gomory war der erste, der die Verwendung von Cutting Planes zur Lösung der ganzzahligen linearen Programmierung vorgeschlagen hat [6]. Die Gomory Cuts, die er vorgestellt hat, werden aus dem gelösten Simplextableau der Relaxierung abgeleitet. Im weiteren Verlauf wird die Herleitung eines Gomory Cut anhand des Beispiels (9.3) durchgeführt. In Abbildung 9.6 wurden die Bedingungen und die Zielgleichung des Problems (9.3) mit zwei zusätzlichen Schlupfvariablen in das Simplextableau überführt und mit dem Simplex-Verfahren gelöst.

Basisvariable	x_1	x_2	s_1	s_2	b
s_1	4	3	1	0	23
s_2	−2	3	0	1	3
Ziel	1	2	0	0	0

(9.6)

Basisvariable	x_1	x_2	s_1	s_2	b
s_1	1	0	$\frac{1}{6}$	$-\frac{1}{6}$	$3\frac{1}{3}$
s_2	0	1	$\frac{1}{9}$	$\frac{2}{9}$	$3\frac{2}{9}$
Ziel	0	0	$-\frac{7}{18}$	$-\frac{5}{18}$	$-9\frac{7}{9}$

(9.7)

Abb. 9.6 Simplextableau zu Beispiel (9.3). Links vor und rechts nach Anwendung des Simplex-Verfahrens.

Aus einer Zeile des gelösten Simplextableaus kann mit der Formel (9.8) ein Gomory Cut abgeleitet werden. [5]

$$\sum_{j=0}^{n}(a_{ij} - \lfloor a_{ij} \rfloor)x_{ij} \geq b_i - \lfloor b_i \rfloor \tag{9.8}$$

Für die zweite Zeile des Simplextableaus ergibt sich folgende Rechnung:

$$(0-0)x_0 + (1-1)x_1 + (\frac{1}{9} - 0)s_1 + (\frac{2}{9} - 0)s_2 \geq (3\frac{2}{9} - 3)$$
$$\frac{1}{9}s_1 + \frac{2}{9}s_2 \geq \frac{2}{9}$$

Es wird somit nur der ganzzahlige Teil vom gebrochenen Teil der Koeffizienten abgezogen, um einen Gomory Cut zu erhalten. Dieser Gomory Cut kann durch Ersetzen der Schlupfvariablen in eine Form der ursprünglichen Variablen gebracht werden.

$$\frac{1}{9}(23 - 4x_1 - 3x_2) + \frac{2}{9}(3 + 2x_1 - 3x_2) \geq \frac{2}{9}$$
$$\frac{29}{9} + 0x_1 - x_2 \geq \frac{2}{9}$$
$$-x_2 \geq -3$$
$$x_2 \leq 3$$

Der Gomory Cute $x_2 \leq 3$ ist in Abbildung 9.5 als blaue Linie grafisch dargestellt. An der grafischen Darstellung wird deutlich, dass dieser schwächer ist als eine optimale Cutting Plane. Um eine weitere bessere Cutting Plane zu erhalten, kann

das Verfahren zur Herleitung des Gomory Cuts auf das neue Problem mit den zwei alten Bedingungen und der ermittelten Cutting Plane erneut angewendet werden. Somit können Schritt für Schritt weitere Cutting Planes hergeleitet werden. Ein eigenständiger Cutting Plane Algorithmus würde genau dies tun, indem es solange das Verfahren wiederholt, bis die Lösung der Relaxierung ganzzahlig ist [6]. Durch die schlechte Güte der Gomory Cuts kann dies jedoch sehr lange dauern. Daher kann es sinnvoll sein, Verfahren zu benutzen die Gomory Cuts verbessern. (z.B. Verfahren aus [4])

9.3.4 Branch-and-Cut

Die oben eingeführten Cutting Planes können auch zur Verbesserung anderer Verfahren beisteuern. Dies wird z.B. beim Branch-and-Cut Algorithmus genutzt, indem ein einfacher Branch-and-Bound Ansatz um Cutting Planes ergänzt wird. Dies kann zu einer signifikanten Beschleunigung von Branch-and-Bound führen [6]. Beim einfachen Branch-and-Bound Ansatz ist das einzige Mittel Teilprobleme einzugrenzen die untere Schranke, mit deren Hilfe Lösungen entweder verworfen werden oder zur neuen unteren Schranke erklärt werden. In allen anderen Fällen muss weiter geteilt werden und so können sehr viele Teilungen notwendig sein, bis das Optimum gefunden ist. Daher ist neben der unteren Schranke eine weitere Möglichkeit gewünscht, den Lösungsraum einzugrenzen und folglich die Anzahl der nötigen Teilungen zu reduzieren.

Die Verwendung von Cutting Planes ist deshalb naheliegend. Denn diese führen dazu, dass der Lösungsraum der Relaxierung besser den Lösungsraum des ganzzahligen Problems approximiert [6], wodurch Teilungen schneller ganzzahlige Lösungen erreichen. Teilprobleme mit ganzzahligen Lösungen müssen nicht weiter geteilt werden und führen zu einer verbesserten unteren Schranke, die wiederum die Anzahl der Teilungen reduziert.

Bei der Verwendung von Cutting Planes im Branch-and-Cut stellt sich jedoch die Frage, wann und wie oft diese gesucht werden. Denn eine Cutting Plane, die für ein Teilproblem gefunden worden ist, gilt nicht für alle anderen Teilprobleme, sondern nur für dieses Teilproblem und alle Teilprobleme, die direkt oder indirekt aus diesem Teilproblem erzeugt worden sind [6]. Dadurch müssen für unterschiedliche Teilprobleme unterschiedliche Cutting Planes gesucht werden. Die Suche mehrerer Cutting Planes für sämtliche Teilprobleme ist daher mit viel Aufwand verbunden. Übermäßiger und unbedachter Einsatz von Cutting Planes kann somit auch zu einer Verlangsamung des gesamten Branch-and-Cut Algorithmus führen.

Sinnvoller ist es, nicht für jedes Teilproblem Cutting Planes zu suchen. Manche Varianten des Algorithmus suchen beispielsweise nur bei jedem achten Teilproblem nach Cutting Planes [6]. Eine andere Möglichkeit ist die *Cut-and-Branch* Variante, bei der nur für das Ursprungsproblem Cutting Planes gesucht werden. Dies hat den Vorteil, dass die Cutting Planes für alle erzeugten Teilprobleme gültig sind [6].

Letztendlich müssen verschiedene Ansätze für die betrachtete Problemklasse getestet werden, um einen effizienten Algorithmus zu konstruieren.

9.4 Fazit

In der Seminararbeit wurden verschiedene Verfahren vorgestellt, Probleme der ganzzahligen linearen Programmierung zu lösen. Die vorgestellten Lösungsstrategien nutzen dabei Verfahren der linearen Programmierung, wie das Simplex-Verfahren, um obere Schranken, untere Schranken und Cutting Planes zu ermitteln und mit diesen den Lösungsraum einzugrenzen. Es wurde verdeutlicht, wie Cutting Planes mit Branch-and-Bound zum Branch-and-Cut Algorithmus kombiniert werden können, um einen schnelleren Algorithmus zu erhalten. Dies setzt jedoch eine sinnvolle Implementierung voraus, bei der Problemwahl, Variablenwahl und Einsatz von Cutting Planes auf das vorliegende Problem zugeschnitten sind.

Vor allem dieser Punkt bietet im Einsatz der Eingenverbrauchsoptimierung weiteres Forschungspotenzial. So ist es sinnvoll zu überprüfen, inwiefern ein Branch-and-Cut Algorithmus implementiert werden muss, um ein effizientes Lösen von Problemen der Eigenverbrauchsoptimierung zu lösen. In diesem Rahmen kann auch überprüft werden, ob problemspezifische Cutting Planes für die vorliegende Problemklasse existieren und inwiefern diese zu einer Beschleunigung des Lösungsverfahren beitragen können. Ebenfalls sollte überprüft werden, wie schnell Heuristiken im Vergleich zur ganzzahligen Programmierung für die vorliegende Problemklasse sind und ob die Güte der Ergebnisse von Heuristiken ausreichend ist.

Literaturverzeichnis

1. Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H.: Branch-and-price: Column generation for solving huge integer programs. Operations Research **46**, 316 – 329 (1996)
2. Bocklisch, T.: Intelligente dezentrale energiespeichersysteme. uwf UmweltWirtschaftsForum **22**(1), 63 – 70 (2014)
3. Kallrath, J.: Gemischt-Ganzzahlige Optimierung: Modellierung in Der Praxis: Mit Fallstudien Aus Chemie, Energiewirtschaft, Metallgewerbe, Produktion Und Logistik. Studium und Praxis. Vieweg+teubner Verlag (2002)
4. Letchford, A.N., Lodi, A.: Strengthening chvátal – gomory cuts and gomory fractional cuts. Operations Research Letters **30**(2), 74 – 82 (2002)
5. Maculan, N.: Integer programming. Handbook of Applied Optimization p. 431 – 445 (2002)
6. Mitchell, J.E.: Branch-and-cut algorithms for combinatorial optimization problems. Handbook of Applied Optimization pp. 65 – 77 (2002)
7. Papadimitriou, C.H.: On the complexity of integer programming. Journal of the ACM **28**(4), 765 – 768 (1981)

Kapitel 10
Projektmanagement

Claas Martin

Zusammenfassung In dem vorliegendem Text wird ein kurzer Überblick über das Themengebiet Projektmanagement gegeben. Auch heutzutage werden viele Projekte nicht erfolgreich abgeschlossen. Im ersten Abschnitt wird gezeigt, warum dies passiert und welche Faktoren daran Schuld sind. Anschließend werde nach einer Definition des Begriffes „Projekt“ die einzelnen Phasen näher beleuchtet, die ein Projekt durchläuft. Darauf aufbauend werden die Aufgaben und Anforderungen des Projektleiters näher betrachtet, der während des Projektes eine Schlüsselrolle einnimmt. Der letzte Abschnitt widmet sich den Vorgehensmodellen. Auch hier wird zunächst ein Überblick geschaffen, bevor zwei Modelle beispielhaft vorgestellt werden. In dieser Ausarbeitung wurde sich auf die zeitliche Sicht auf ein Projekt beschränkt und Kosten- und Ressourcenplanung etwas außen vor gelassen. Dies ist für unsere Projektgruppe nicht relevant, da es kein finanzielles Budget gibt.

10.1 Motivation

Seit Jahren wird in Unternehmen vermehrt Informationstechnologie (IT) eingesetzt, die Investitionen in IT steigen [1]. Dem entsprechend wird es in Zukunft eine große Anzahl an IT-Projekten geben, die durchgeführt werden müssen. In Abbildung 10.1 ist jedoch erkennbar, dass der Anteil der erfolgreich abgeschlossenen Projekte seit Jahren zwischen 30% und 40% pendelt. Auch der Anteil der Projekte, die nicht innerhalb des Zeit- oder Kostenrahmens bzw. die gescheitert sind, hat sich nicht signifikant verändert.

Projektmanagement ist ein alter Prozess. Bereits seit Mitte des 20. Jahrhunderts wurde es bei großen Projekten, wie dem Bau der transkontinentalen Eisenbahnlinie in den USA, eingesetzt [5]. Das wirft die Frage auf, warum auch heutzutage nur so wenige Projekte erfolgreich abgeschlossen werden können. Eine Studie aus dem

Carl von Ossietzky Universität Oldenburg
E-mail: claas.martin@uni-oldenburg.de

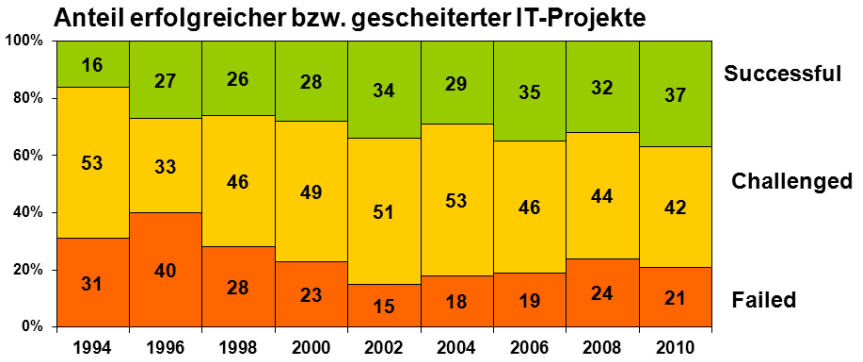


Abb. 10.1 Anteil erfolgreicher bzw. gescheiterter IT-Projekte. [11]

Jahr 2008 zeigt auf, dass vor allem mangelnde Kommunikation ein großes Problem darstellt (Abbildung 10.2). Für das Gelingen eines Projektes ist die Komplexität der Aufgabenstellung nicht so entscheidend, wie es klare Zielvorgaben sind. Im Abschnitt 10.2.3 wird gezeigt, was genau unter „klaren“ Zielen verstanden wird.

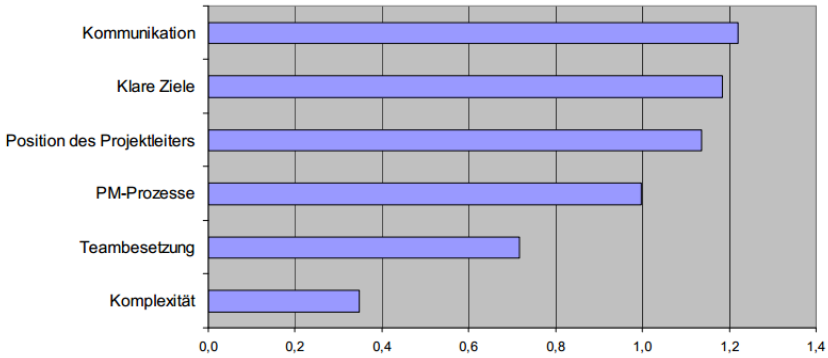


Abb. 10.2 Anteil erfolgreicher bzw. gescheiterter IT-Projekte. [2]

10.2 Projekt

Projekte begegnen uns im Alltag in verschiedenen Situationen, ohne dass wir sie konkret als Projekte bezeichnen. Intuitiv würden viele wohl sagen, dass Projekte „große“, einmalige (Weiter)-Entwicklungen sind. Beispiele sind der Bau eines Ei-

genheimes oder der Abschluss eines Studiums. In öffentlichen Bereichen begegnen wir Projekten beim Bau von Straßen oder Häfen oder im gesellschaftlichen Bereich zum Beispiel bei der Durchführung einer Wahl.

Projekte sind in Unternehmen ein wichtiger Bestandteil erfolgreichen Wirtschaftens, da sie Weiterentwicklungen ermöglichen. Es gibt technische Projekte, bei der neue Maschinen entwickelt und im Unternehmen eingeführt werden. Daneben gibt es betriebswirtschaftliche Projekte, bei der vielleicht Prozesse verändert werden, Unternehmen fusionieren oder neu gegründet werden [6].

IT-Projekte fallen in den Bereich betriebswirtschaftlicher Projekte. In diesem Bereich ist weiterhin mit steigenden Investitionen zu rechnen [1]. Projekte werden sowohl von Unternehmen durchgeführt, die Software oder Hardware entwickeln, als auch in Unternehmen, die diese Produkte anschließend einführen. Die Anzahl durchzuführender Projekte wird sich dem zufolge in gleichem Maße weiter erhöhen. Da es so viele Projekte gibt, bietet es sich an, Projektmanagement einzusetzen, um die Projekte erfolgreich abzuschließen.

10.2.1 Definition

> Vorhaben, das im Wesentlichen durch die Einmaligkeit aber auch Konstante der Bedingungen in ihrer Gesamtheit gekennzeichnet ist, wie z. B. Zielvorgabe, zeitliche, finanzielle, personelle und andere Begrenzungen; Abgrenzung gegenüber anderen Vorhaben; projektspezifische Organisation. (DIN 699001)

Eine Definition für alle Projekte abzugeben ist schwer, da Projekte in unterschiedlichsten Bereichen, Größendimensionen und mit verschiedenen Zielsetzungen anzutreffen sind. Einige Punkte können jedoch auf alle Projekte bezogen werden. Der Name „Projekt“ stammt aus dem lateinischen vom Wort *proiectum* ab. Wörtlich übersetzt bedeutet dies „das nach vorn geworfene“ [10]. Ein Projekt bringt etwas nach vorne, im Mittelpunkt eines Projektes steht deshalb ein innovatives Ziel oder Aufgabenstellung. Dieses Ziel kann in Teilprojekte zerlegt werden. Projekte werden innerhalb eines vorher festgelegten Zeitrahmens durchgeführt, sie besitzen einen Start- und Endzeitpunkt. Hinzu kommt, dass Projekte einmalig durch ein Team durchgeführt werden, dem nur gewissen Ressourcen zur Verfügung stehen [3] [6] [9] [10].

10.2.2 Projektphasen

Projekte können grob in vier unterschiedliche Phasen eingeteilt werden [7], die [10] Think, Plan, Build und Run nennt. Ein Projekt beginnt mit der *Think*-Phase, in der die Projektdefinition stattfindet. In dieser Phase wird das Projektziel und eine Grobplanung festgelegt, sowie das Projekt organisiert. Das bedeutet zum Beispiel, dass das Team zusammen gestellt wird und die Rollen definiert werden.

Die nächste Phase ist die *Plan*-Phase. Hier findet die Feinplanung des Projektes statt. Dies Umfasst eine genauere Planung von Arbeitspaketen, Termine die eingehalten werden müssen und Kosten- und Ressourcenschätzungen. Daraus können Projektpläne abgeleitet werden, die zum Beispiel den zeitlichen Verlauf oder Abhängigkeiten verschiedener Arbeitspakete darstellen.

In der *Build*-Phase wird das Projekt realisiert. Wichtige Schritte sind hier die Kontrolle und Steuerung der durchgeführten Arbeit. Kontrolliert werden muss, ob die einzelnen Arbeitspakete fachgerecht durchgeführt werden und ob sie noch in der Zeit- und Kostenplanung liegen. Werden Abweichungen von der Planung festgestellt, müssen Steuerungsmaßnahmen durchgeführt werden. Möglich ist zum Beispiel eine Verschiebung des Fertigstellungstermins, das zuweisen von mehr Mitarbeitern oder eine Überarbeitung der Anforderungen. Ebenfalls fallen Qualitätssicherungsmaßnahmen in diese Phase des Projektes.

Die letzte Phase ist die *Run*-Phase, in der das Projekt abgeschlossen wird. Das Produkt wird vom Auftraggeber abgenommen. Neben dem Projekt wird auch der Projektabschlussbericht vorgelegt. Ist die Übergabe erfolgt, findet im Team eine Auswertung des Projektes statt. Dabei sollen Erfahrungen gesichert werden, um darauf später zugreifen zu können. Anschließend wird das Team aufgelöst und wieder in das Unternehmen integriert.

10.2.3 Ziele und Aufgaben

Am Ende eines Projektes soll ein konkretes Ziel erreicht worden sein. Dieses Hauptziel wird im Verlauf eines Projektes in viele kleine Teilziele zerlegt, um daraus konkrete Aufgaben zu generieren. Diese werden einzelnen Mitgliedern oder Gruppen zugewiesen und von ihnen bearbeitet. Häufig werden Ziele mit der *S.M.A.R.T.*-Formel erstellt. Die Ziele müssen dabei die fünf Kriterien spezifisch, messbar, akzeptiert, realistisch und terminierend sein [10].

Im Kontext der Zieldefinition meint *spezifisch*, dass das Ziel so detailliert wie möglich und auf das Projekt bezogen formuliert werden muss. *Messbar* bedeutet, dass Ziele überprüfbar sein müssen. Dazu ist eine konkrete Angabe notwendig, was erreicht werden soll. Eine schlecht messbare Aussage ist zum Beispiel: „Ich möchte mehr Sport machen“. Besser ist die Formulierung: „Ich fahre zweimal in der Woche mit dem Fahrrad zur Arbeit“. Für das Projekt sind solche klar und detaillierten Formulierungen wichtig, um den Projektfortschritt zu kontrollieren. Es kann schnell entschieden werden, ob ein Ziel erreicht wurde oder nicht.

Damit alle Mitglieder des Projektteams hinter dem Ziel stehen, muss das Ziel *akzeptiert* werden. Ebenso müssen die Ziele realistisch sein. Das bedeutet zum Beispiel auch, dass der Bearbeiter der Aufgabe dieser auch gewachsen ist, indem das Ziel nicht deutlich über seinen Kenntnisstand hinausgeht oder es in der vorgesehenen Zeit bearbeitet werden kann. Dafür besitzt jedes Ziel einen definierten Start- und Endpunkt. Dies ist der letzte Punkt in der SMART-Formel, das Ziel *terminiert*.

Das Projekt gliedert sich auf den obersten Ebenen und das Projektziel und davon ausgehend in verschiedene Teilziele. Diese Teilziele werden wiederum in Ergebnisziele unterteilt, aus denen die Arbeitspakete erstellt werden. Arbeitspakete können nach DIN 69901 nicht weiter aufgegliedert werden [6]. Sind die Aufgaben für das Projekt definiert, werden sie geordnet. Dies gescheit mit der Projektstrukturplanung.

10.2.4 Strukturplanung

Vor der Strukturplanung wurde durch die Anforderungsdefinition die Frage geklärt, *was* in dem Projekt gemacht werden soll. In der Strukturplanung geht es nun darum, weitere Fragen zu den Arbeitspakten zu klären. Dazu gehört, *wer* eine bestimmte Aufgabe übernimmt. Es ist möglich, dass auch Aufgaben an externe Partner vergeben werden. Eine weitere Frage ist, *wann* eine Aufgabe ausgeführt wird. Hier muss auf Abhängigkeiten von verschiedenen Paketen geachtet werden. Kann ein Vorgang erst nach einem anderen durchgeführt werden, muss eine entsprechende Reihenfolge festgelegt werden. Ebenso kann festgesetzt werden, *wie*, mit welcher Arbeitsmethode, und *womit*, mit welchen Tools, eine Aufgabe bearbeitet werden soll. Zur Kostenschätzung gehört in das Arbeitspaket ebenfalls eine Schätzung *wie viel* es kostet [6].

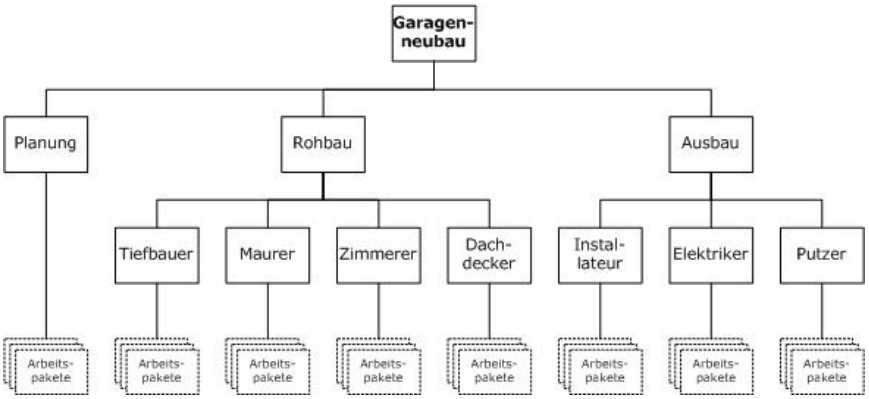


Abb. 10.3 Ein Projektstrukturplan am Beispiel eines Garagenneubaus. [8]

Sind die Arbeitspakete fertig definiert, erstellt man einen Strukturplan wie in Abbildung 10.3 dargestellt. Unter dem Projektziel, die Garage neu zu bauen, werden verschiedene Teilziele erstellt, die anschließend in Arbeitspaketen münden. Auf der obersten Ebene der Teilziele ist eine zeitliche Sortierung vorgenommen worden, da zuerst die Planung und anschließend der Rohbau und Ausbau ausgeführt werden

müssen. Die untere Ebene der Teilziele ist funktional nach den einzelnen Subunternehmen gegliedert.

10.2.5 Ablaufplanung

Durch die Strukturplanung erhält man einen guten Überblick über das Projekt. Es stellt jedoch keine zeitlichen Abläufe oder Abhängigkeiten von Arbeitspaketen untereinander dar. Dafür ist die Darstellung als Gantt-Diagramm geeignet. Es ist ein Balkendiagramm, in dem auf der x-Achse die Zeit und auf der y-Achse die Arbeitspakete dargestellt werden. Neben den Arbeitspaketen können auch Phasen und Meilensteine integriert werden. Die Länge der Balken ist proportional zur geplanten Zeit, die für das Paket zur Verfügung steht [6]. Jeder Vorgang, somit auch jeder Balken, wird in einer Zeile dargestellt, sodass parallel laufende Vorgänge einem Überlappen der Balken entspricht.

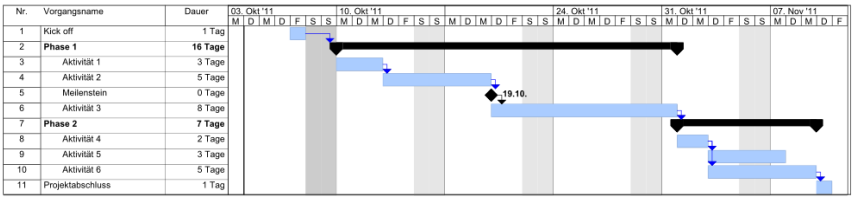


Abb. 10.4 Beispiel eines Gantt-Diagramms. Balken, die durch die blauen Pfeile verbunden sind, bilden den kritischen Pfad.

In einem Gantt-Diagramm kann auch der kritische Pfad eingezeichnet werden, wie in Abbildung 10.4 zu sehen ist. Dieser Pfad verbindet die Elemente, bei denen eine Verzögerung eines einzelnen Vorgangs zu einer Verzögerung des ganzen Projektes führt. Andere Vorgänge, die nicht auf dem kritischen Pfad liegen, haben eine gewisse Pufferzeit. Ihr Endtermin kann nach hinten geschoben werden, ohne dass sich darauf aufbauende Vorgänge verzögern. In einem Gantt-Diagramm werden zeitliche Beziehungen übersichtlich dargestellt. Andere Bedingungen, wie zum Beispiel die Abhängigkeit bestimmter Ressourcen, können damit jedoch nicht visualisiert werden.

10.2.6 Controlling

Nachdem die Planung des Projektes abgeschlossen ist, beginnt in der *Build*-Phase die Realisierung des Projektes. Die Arbeitspakete werden nun der Reihenfolge entsprechend abgearbeitet. In dieser Phase des Projektes sind die Hauptaufgaben des

Managements die Kontrolle und Steuerung des Projektes. Der aktuelle Projektfortschritt wird mit der Planung verglichen. Werden Abweichungen sichtbar, muss darauf entsprechend reagiert werden.

Der Prozess des Controllings umfasst jedoch mehr als die Erkennung und Behandlung von Termin- oder Kostenabweichungen [6]. Nach DIN 69904 umfasst Controlling alle „Prozesse und Regeln, die innerhalb des Projektmanagements zur Sicherung des Erreichens des Projektziels beitragen“ [10]. Das bedeutet auch, dass die Planung nicht starr sind, sondern im Verlauf eines Projektes verändert werden. Dies kann zum Beispiel durch sich ändernde Anforderungen oder Termine geschehen oder dadurch das Maßnahmen ergriffen werden müssen, um die Abweichung zu minimieren.

10.2.7 Projektleiter

Dem Projektleiter kommt während des Projektes eine besondere Rolle zu. Seine Hauptaufgabe ist es, das Projekt zum Erfolg zu führen. Dabei muss er auf die Anforderungen des Kunden eingehen und darf die Bedürfnisse seines Teams nicht außer acht lassen. Abbildung 10.5 zeigt den Rahmen, in dem sich ein Projektleiter bewegt. Ein Projekt wird im Idealfall im Zeit- und Kostenrahmen mit den gewünschten Funktionen realisiert. Diese drei Eigenschaften verhalten sich jedoch gegensätzlich. Das bedeutet, gibt es Abweichungen in der Realisation zur Planung, können in der Regel nur zwei der drei Eigenschaften des Projektes aufrechterhalten werden. Liegt das Projekt zum Beispiel hinter dem Zeitplan, der allerdings mit der gewünschten Funktionalität gehalten werden soll, können weitere Entwickler eingestellt werden, die die Arbeit erledigen. Dies führt allerdings zu höheren Kosten, sodass dieses Ziel nicht erreicht wird.

Zu Beginn eines Projektes, wenn das Team frisch zusammengestellt worden ist, gibt es noch eine hohe Abhängigkeit vom Projektleiter. Er muss das Team führen und gibt alle zu verrichtenden Arbeiten vor. Mit zunehmender Projektdauer wachsen die einzelnen Mitglieder des Projektes zu einem Team zusammen. Es gibt Konflikte, die durch das Projektmanagement aufgelöst werden müssen, und Kompromisse werden geschlossen. Gleichzeitig festigen sich die Rollen, die von den Mitarbeitern in dem Projekt ausgefüllt werden. In dieser Phase werden bereits Aufgaben an das Team delegiert. Die Entscheidungsbefugnis, wie ein Vorgang konkret bearbeitet wird, bleibt jedoch bei der Projektleitung. In weiteren Phasen unterstützt die Leitung das Team weiterhin, delegiert jedoch immer mehr Aufgaben. Durch gemeinsam festgelegte Regeln und erfolgreich bewältigte Probleme entwickelt sich ein Team, welches an einer gemeinsamen Vision arbeitet. Dies führt schließlich dazu, dass das Team Probleme eigenständig löst. [10] bringt diesen Entwicklungsprozess auf die einfache Formel: „Ein Team leiten heißt, andere Erfolg haben lassen.“

Der Projektleiter muss allerdings nicht nur das Team intern betreuen, es muss das Projekt auch nach außen vertreten. Er ist der Ansprechpartner für den Auftraggeber. Dies beginnt bei der Auftragsvergabe, bei der er die Verhandlungen mit gestaltet.

Abb. 10.5 Spannungsdreieck des Projektmanagements

Er übernimmt „die Planung, Steuerung und Überwachung des Projektes“ (vgl DIN 69901), sodass er auch auf wechselnde Anforderungen des Kunden oder Probleme bei der Realisierung mit dem Auftraggeber in Kontakt stehen muss. Am Ende muss er das Projekt abschließen.

10.3 Vorgehensmodell

In den letzten Jahren wurden viele Vorgehensmodelle entwickelt. Dies liegt auch daran, dass es viele verschiedene Arten von Projekten gibt, deren Ziele und Projektgröße sehr stark variiert [6]. Allen Vorgehensmodellen gemein ist, dass sie verschiedene Phasen definieren, in denen ähnliche Aufgaben zusammengefasst werden. Wie oft und welcher Reihenfolge solche Phasen durchlaufen werden, variiert jedoch sehr stark.

Bei sequentiellen Vorgehensmodellen werden die einzelnen Phasen festgelegt und der Reihe nach abgearbeitet. Abbildung 10.6 zeigt solch ein Vorgehensmodell mit fünf Phasen. Um in eine neue Phase einzutreten, muss die vorhergehende Phase komplett abgeschlossen sein. Vorteile von sequentiellen Vorgehensmodellen sind unter anderem, dass sie klar strukturiert und überschaubar sind. Der Managementaufwand ist dadurch gering und die Modell sind leicht zu verstehen [4]. Nachteilig wirkt sich hingegen aus, dass nicht auf sich verändernde Anforderungen eingegangen werden kann. Zudem wird das entwickelte System nur am Ende getestet. Stellt sich dabei heraus, dass das Produkt nicht dem Kundenwunsch entspricht, kann dies nicht verändert werden. Die Modelle sind recht starr. Eingesetzt werden können solche

Vorgehensmodelle bei kleinen Projekten und wenn davon ausgegangen werden kann, dass sich die Anforderungen während des Projektes nicht ändern [6].

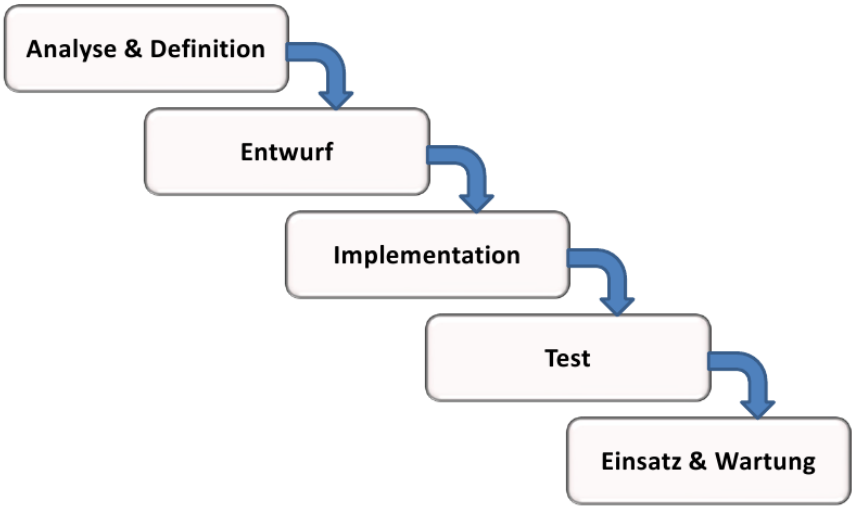


Abb. 10.6 Einfaches sequentielles Vorgehensmodell mit den fünf Phasen „Analyse und Definition“, „Entwurf“, „Implementation“, „Test“, „Einsatz und Wartung“. [4]

Um den Nachteil der sequentiellen Modelle, dass die Risiken erst am Ende des Projektes sichtbar werden, zu begegnen, wurden iterative Verfahren entwickelt. Hierbei wird das zu entwickelnde System in Teilsysteme zerlegt, die einzeln entwickelt werden können [6]. Die Phasen werden mehrmals wiederholt und nach jedem Phasendurchlauf entsteht ein Produkt, welches ein Teil dem bereits Entwickelten hinzufügt. Durch die früh lauffähigen Systeme bekommt der Auftraggeber früh einen Eindruck des Produktes und kann auf etwaige Fehlentwicklungen frühzeitig Einfluss nehmen. Dadurch sind sich verändernde Anforderungen jedoch eher die Regel. Ein weiteres Problem ist, dass die Systemarchitektur relativ früh festgelegt wird. Wird später festgestellt, dass die Architektur die Anforderungen nicht erfüllt, ist der Aufwand die Änderungen durchzuführen relativ hoch [6].

Heutzutage sind agile Vorgehensmodelle ebenfalls weit verbreitet. Sie sind besonders geeignet, wenn unklare Anforderungen und sich verändernde Ziele vorliegen. Im Vergleich mit den anderen Vorgehensweisen weisen sie den größten Managementaufwand auf. Sie bauen auch auf iterativen Modellen auf. Die Entwicklungszyklen sind hier jedoch sehr kurz gewählt, sodass sehr schnell eine lauffähige Version vorliegt, die noch nicht alle Projektziele umfassen muss. Durch die am Anfang unklaren Anforderungen und dem Drang nach schnellen lauffähigen Systemen, kann man zu Beginn noch nicht genau sagen, wo das Projekt am Ende stehen wird. Ebenfalls

gibt es bei agilen Vorgehensmodellen keine Qualitätsgarantie, da die Dokumentation aufgrund der wechselnden Anforderungen möglichst gering gehalten wird.

10.3.1 V-Modell XT

Das V-Modell ist ein 1995 vorgestelltes Vorgehensmodell. Die Entwicklung wurde von der Bundesrepublik Deutschland gefördert und ist nun das Standardvorgehensmodell für IT-Projekte der öffentlichen Hand. Im Mittelpunkt stehen *Produkte*. Diese sind die Ergebnisse der einzelnen Phasen und sind nicht nur das Softwareprodukt, sondern auch alle Dokumente, die zum Projekt gehören. Für die einzelnen Phasen gibt es detaillierte Vorgeben, welche Rolle in einem Projekt einzelne Aufgaben zu erledigen hat, bzw. welche Produkte am Ende der Phase vorhanden sein müssen. Es legt somit fest, *wer wann was* macht. Nicht festgelegt wird hingegen, wie das Produkt erstellt wird. Damit lässt sich das Vorgehensmodell mit weiteren Vorgehen kreuzen. Dafür steht auch die Abkürzung XT (engl. extreme tailoring), die die Anpassbarkeit des Vorgehensmodell an unternehmensspezifische Gewohnheiten oder Bedürfnisse hervorstellt.

10.3.2 SCRUM

SCRUM baut auf einem iterativen, inkrementellen Ansatz auf. Die Beobachtung, nach der SCRUM entwickelt wurde war, dass heutige Projekte sehr groß und komplex werden können, sodass sie nicht am Beginn vollständig geplant werden können. Es reicht nicht, für ein komplexes Projekt, einen komplexen Plan zu erstellen. SCRUM versucht, die Komplexität des Projektes leicht zu verwalten. Dazu gibt es drei Kernbausteine. Zum ersten wird *Transparenz* gefordert. Der Fortschritt bzw. Hindernisse werden für alle sichtbar festgehalten. Als zweites sollen in regelmäßigen Abschnitten die Produktfunktionalitäten *überprüft* werden. Als drittes werden die Anforderungen nach jedem Iterationsschritt neu bewertet und entsprechend *angepasst*.

Bei SCRUM steht auch eine schnelle erste Lieferung eines lauffähigen Produktes im Vordergrund, die anschließend in weiteren Schritten um Funktionalität erweitert wird.

Literaturverzeichnis

1. Borbe, A.: Weltweite IT-Ausgaben sollen 2014 wieder steigen. Website (2014). URL `http://www.silicon.de/41593613/weltweite-it-ausgaben-sollen-2014-wieder-steigen/`. (abgerufen 30.05.2014)

2. Engel, C., Tamdjidi, A., Quadejacob, N.: Ergebnisse der Projektmanagement Studie 2008 - Erfolg und Scheitern im Projektmanagement. Präsentation (2008). http://www.gpm-ipma.de/fileadmin/user_upload/Know-How/Ergebnisse_Erfolg_und_Scheitern-Studie_2008.pdf (zuletzt abgerufen 11.06.2014)
3. Fischer, D.I.F.: Projektmanagement, 7. auflage edn. Prof. Dr. Anke Hanft (2012)
4. Koord, Y., Krauter, V.: Überblick Vorgehensmodelle im Projektmanagement. Wiki (2009). URL winfwiki.wi-forum.de/index.php/Überblick_Vorgehensmodelle_im_Projektmanagement. (zuletzt abgerufen: 14.06.2014)
5. Microsoft: Eine kurze Geschichte des Projektmanagements. Website (2014). URL http://office.microsoft.com/de-de/project-help/eine-kurze-geschichte-des-projektmanagements-HA001135342.aspx. (zuletzt abgerufen 11.06.2014)
6. Ruf, W., Fittkau, T.: Ganzheitliches IT-Projektmanagement. Oldenbourg Verlag (2008)
7. Sauer, J.: IT-Projektmanagement. Vorlesungsfolien (2012)
8. Seelhöfer, M.: Bild (2013). URL upload.wikimedia.org/wikipedia/de/1/1e/PSP_Garage.jpg. (zuletzt abgerufen 14.06.2014)
9. Voigt, K.I., Schewe, G.: Stichwort: Projekt. Gabler Wirtschaftslexikon. URL http://wirtschaftslexikon.gabler.de/Archiv/13507/projekt-v7.html. Version 7
10. Wilkeit, E.: Soft Skills - (nicht nur) für Informatikerinnen und Informatiker. Vorlesungsskript (2011)
11. Zank, I.: Sensibilität, Virtualität und Variabilität - Woran IT-Projekte scheitern ! Website (2009). URL http://www.ikmt.de/forum/showthread.php?tid=334&pid=462#pid462. (zuletzt abgerufen 11.06.2014)

Kapitel 11
Requirements Engineering

Christian Best

Zusammenfassung In dieser Ausarbeitung wird der Grundstein für das Requirement Engineering in einer Projektgruppe gelegt. Deshalb wird zunächst in Abschnitt 11.2 definiert, was unter einer Anforderung und unter Requirements Engineering verstanden wird, um ein gemeinsames Verständnis zu erreichen. Nach dieser Begriffsdefinition wird in Abschnitt 11.3 darauf eingegangen, warum Anforderungen sinnvoll sind. Der Sinn von Anforderungen wird durch die verschiedenen Funktionen erläutert, die Anforderungen erfüllen. Nachdem damit das Was und Warum geklärt ist, wird in Abschnitt 11.4 das Wie erläutert. Wie können Anforderungen sinnvoll notiert werden? Bevor mit Anforderungen gearbeitet werden kann, müssen diese erhoben werden. Wie eine solche Erhebung aussehen kann, wird in Abschnitt 11.5 dargestellt. Das Requirements Engineering steht allerdings nicht alleine, sondern ist in einem Kontext zu verstehen. Deshalb wird in Abschnitt 11.6 auf Anforderungen im Projektalltag eingegangen und der Bezug zu anderen Managementaufgaben in einem Projekt hergestellt. Abschließend wird in Abschnitt 11.7 ein Fazit der Arbeit und ein Ausblick auf offene Fragen gegeben.

11.1 Einführung

Die Beschäftigung mit Anforderungen ist kein Selbstzweck, sondern dem Umstand geschuldet, dass viele Projekte auf Grund von ungenügenden Anforderungen scheitern. Laut einer Studie der Standish Group scheitern 13% der Projekte wegen unvollständigen Anforderungen. Darüber hinaus scheitern weitere Projekte aus Gründen, die mit besseren Anforderungen bzw. Anforderungsmanagement nicht aufgetreten wären, wie fehlendes Benutzerinteresse, fehlende Ressourcen, unrealistische Erwartungen oder sich verändernde Anforderungen. [6, S. 10] Anforderungen sind somit ein wichtiger Grundstein für Projekte.

Carl von Ossietzky Universität Oldenburg
E-mail: christian.best@uni-oldenburg.de

Diese Ausarbeitung legt den Grundstein für die Projektgruppe „Hardwarebasierte Simulation energieautonomer Gebäude“. Nach der ersten Vision der Projektgruppe werden auf Industrie PCs Erzeuger, Verbraucher und Speicher simuliert sowie eine Steuerungseinheit ausgeführt. Es handelt sich dem ersten Eindruck nach um ein verteiltes System mit vielfältiger Kommunikation der verschiedenen Komponenten untereinander.

Zum Zeitpunkt dieser Arbeit befindet sich die Gruppe noch im Anfangsstadium und ein Requirements Engineering muss erst noch etabliert werden. Aus diesem Grund konzentriert sich die Arbeit auf die zunächst anstehenden Tätigkeiten der Erhebung und Entwicklung von Anforderungen.

Die Teilnehmer des Projekts haben alle vertiefte Kenntnisse in der Unified Modelling Language und in der Modellierung von Systemen. Das Domänenwissen ist sehr unterschiedlich verteilt. Insbesondere die Kunden bzw. Auftraggeber haben ein sehr vertieftes Wissen, während die Auftragnehmer nur ausschnittsweise Wissen in der Domäne haben.

11.2 Definitionen

In diesem Abschnitt werden die Grundlagen für die weiteren Kapitel gelegt, indem zunächst geklärt wird, was eine Anforderung ist. Dazu wird in dem Abschnitt 11.2.1 eine Definition von Anforderungen gegeben und anschließend wird auf unterschiedliche Anforderungsarten und Merkmale von Anforderungen eingegangen. Danach wird in Abschnitt 11.2.2 erläutert, was unter der zu Anforderungen gehörigen Disziplin, dem Requirements Engineering, zu verstehen ist.

11.2.1 Anforderung

In diesem Abschnitt werden die Grundlagen für die weiteren Kapitel gelegt, indem geklärt wird, was eine Anforderung ist.

Eine gängige Definition von Anforderungen ist die Definition nach IEEE. Diese besagt, dass unter einer Anforderung eine dokumentierte Darstellung von einer Beschaffenheit oder Fähigkeit verstanden wird, die entweder zur Lösung eines Problems des Systemnutzers oder zur Erfüllung eines formellen Dokuments notwendig ist. [8, S. 13] [9, S. 32ff] [3, S. 9]

Auffällig an dieser Definition ist, dass lediglich dokumentierte Darstellungen als Anforderung angesehen werden. Vorstellungen von Kunden werden somit erst dann zu einer Anforderung, wenn sie dokumentiert worden sind. [8, S. 13]

Anforderungen werden für verschiedene Zwecke verwendet. Sie werden für die Abschätzung, Planung, Durchführung und Verfolgung von Projekttätigkeiten benötigt. Dementsprechend können verschiedene Arten von Anforderungen unterschie-

den werden, um sie zu systematisieren. [3, S. 9] Anforderungen können zum einen quer und zum anderen längst geschnitten werden.

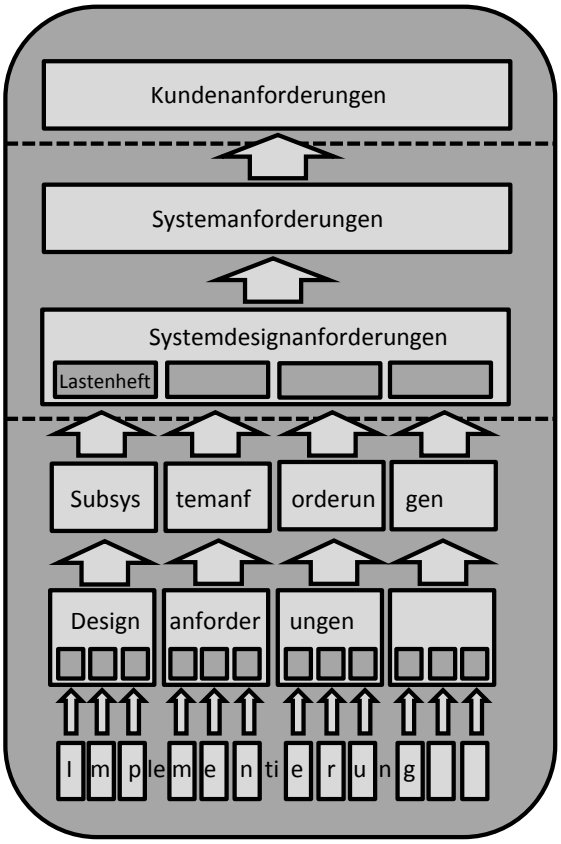


Abb. 11.1 Querschnitt von Anforderungen aus [6, S.63]

In Abbildung 11.1 wird der Querschnitt von Anforderungen visualisiert. Quer bedeutet in diesem Fall nach Implementierungsferne sortiert. Kundenanforderungen sind besonders weit von der Implementierung entfernt, während Designanforderungen in direkter Verbindung zu den Implementierungen stehen. Je nach Quelle finden sich unterschiedliche Möglichkeiten den Querschnitt vorzunehmen. Dabei stimmen oft die ersten drei Ebenen überein, welche geordnet von implementierungsfern zu implementierungsnah Kundenanforderungen, Systemanforderungen und Komponentenanforderung sind. [7, S. 26] [6, S. 63] [1, S. 24] Das Modell in Abbildung 11.1 ist besonders dafür geeignet, um zu visualisieren, wie aus Kundenanforderungen handhabbare Pakete bzw. Implementierungen werden. Durch diese durchgehende

Systematik scheint es besonders geeignet für ein Projekt, das ein erstes Requirements Engineering initialisieren möchte, wie in Abschnitt 11.1 dargestellt.

Der bekannteste Längsschnitt bei Anforderungen ist die Unterscheidung in funktionale und nichtfunktionale Anforderungen. [9, S.125] Darüber hinaus finden sich verschiedene Systematiken in der Literatur. Diese Systematiken unterscheiden sich nicht nur darin, was für Anforderungsarten neben den funktionalen Anforderungen aufgelistet werden, sondern bereits in der Zuordnung von Anforderungsarten zu funktional oder nichtfunktional. So wird die Benutzerschnittstelle in [8] explizit neben die funktionalen Anforderungen gestellt, während sie bei [3] zu den funktionalen Anforderungen gezählt wird. Interessanterweise geht keine der verschiedenen Systematiken auf andere bestehende Systematiken ein oder liefert Begründungen für ihre Systematik. Dies weist darauf hin, dass die Systematik selbst weniger wichtig ist, als das Vorhandensein irgendeiner Systematik. Dies kann dem Umstand geschuldet sein, dass die Aufteilung in Arten der Anforderungen hilft, das Spektrum der Anforderungen im Blick zu behalten und keine Anforderungen zu vergessen. Dafür scheint eine detaillierte Darstellung wie in [8] besonders geeignet, weshalb sie im Folgenden dargelegt wird.

[8] unterscheidet zwischen funktionalen Anforderungen, technischen Anforderungen, Anforderungen an die Benutzerschnittstelle, Qualitätsanforderungen, Anforderungen an sonstige Lieferbestandteile, Anforderungen an die Durchführung der Entwicklung und Einführung sowie rechtlich-vertragliche Anforderungen.

Funktionale Anforderungen werden auch Verhaltensanforderungen genannt, weil sie Anforderungen an das Verhalten des Systems, also die Aktionen, die ein System ausführen kann, beschreibt. Ein Beispiel für eine funktionale Anforderung ist „Falls ein registrierter Gast sich angemeldet hat, soll das Restaurantsystem diesem Gast eine persönliche Begrüßung, seine drei Lieblingsspeisen und eine Empfehlung des Hauses anzeigen.“(s. [8, S. 234]).

Technische Anforderungen hingegen beschreiben Anforderungen an die Hardware, Software oder Programmiersprachen, die in einem System verwendet werden sollen. Ein Beispiel für eine technische Anforderung ist „Das System soll einen mit dem Auftraggeber abgestimmten Open-Source-Browser unterstützen.“(s. [8, S. 261]).

Anforderungen an die Benutzerschnittstelle legen fest, was für Interaktionen mit den Benutzern des Systems möglich sein sollen. Diese Interaktionen können sowohl zwischen dem System und Menschen stattfinden, wie auch zwischen dem System und anderen Systemen. Ein Beispiel für eine Anforderung an die Benutzerschnittstelle ist „Das System soll ausschließlich die in dem Corporate Design des Auftraggebers festgelegten Farbwerte verwenden.“.

Qualitätsanforderungen treffen Vorgaben über die Güte des Produkts, des Prozesses zum Erstellen des Produkts oder den an der Konstruktion des Produkts beteiligten Personen. Ein Beispiel für eine Qualitätsanforderung ist „Falls eine Bearbeitung durch den Benutzer oder durch sonstige Fehler unterbrochen wird, darf dies nicht zu Inkonsistenzen in der Datenbank führen“(s. [8, S. 273]).

Anforderungen an sonstige Lieferbestandteile können Anforderungen an das Benutzerhandbuch oder Anforderungen an das Installationshandbuch sein. Ein Beispiel

für eine Anforderung an sonstige Lieferbestandteile ist „Der Auftragnehmer muss dem Auftraggeber Schulungen in deutscher Sprache für folgende Benutzergruppen anbieten: Kellner, Koch, IT-Support und Restaurantmanager.“(s. [8, S. 277]).

Anforderungen an durchzuführende Tätigkeiten können bestimmen, nach welchem Vorgehensmodell das System entwickelt werden soll, welche Standards eingehalten werden müssen oder wie der Auftraggeber über den Status des Projekts informiert wird. Ein Beispiel für eine Anforderung an durchzuführende Tätigkeiten ist „Der Auftragnehmer soll einen Qualitätsmanagementplan für das Projekt erstellen und vom Auftraggeber genehmigen lassen.“([8, S. 279]).

Rechtlich-vertragliche Anforderungen regeln das Verhalten des Auftraggebers, sowie des Auftragnehmers. Es legt unter anderem fest, wie und wann Zahlungen geleistet werden, wie mit Unstimmigkeiten oder Anforderungsänderungen umgegangen wird. Ein Beispiel für eine rechtlich-vertragliche Anforderung ist „Wenn der Auftragnehmer erkennt, dass Anforderungen des Auftraggebers nicht erfüllbar sind, so hat er dies und die ihm erkennbaren Folgen dem Auftraggeber sofort schriftlich mitzuteilen.“(s. [8, S. 283]).

11.2.2 Requirements Engineering

Sowohl in der deutschsprachigen, wie auch in der englischsprachigen Literatur ist die Abgrenzung zwischen den Begriffen Requirements Engineering und Anforderungsmanagement bzw. Requirements Management nicht eindeutig. Zum einen werden alle mit Anforderungen zusammenhängenden Tätigkeiten als Anforderungsmanagement verstanden [9, S. 32] [1, S. 4] und zum anderen wird Anforderungsmanagement lediglich als das Verwalten von Anforderungen verstanden, das einen Teilbereich des Requirements Engineerings darstellt. [8, S. 14] [11, S. 215]

In dieser Arbeit wird Requirements Engineering als der übergeordnete Begriff verstanden. Dies geht mit dem Begriffsverständnis von [8] einher. Darüber hinaus wurde in [9, S. 32] der Begriff Anforderungsmanagement lediglich bevorzugt, weil mehr Probleme im Management als im Engineering lägen und der Begriff Anforderungsmanagement sich besser an den Mann bringen ließe. Wie [9] selbst argumentiert, ist der Begriff Requirements Engineering der passendere Begriff. Seine beiden Argumente wiegen nicht den Nachteil eines unpassenden Begriffs auf.

Eine Definition des Requirements Engineerings erfolgt am einfachsten über seine Bestandteile, die in Abbildung 11.2 visualisiert werden.

[9] unterscheidet die operationalen Aufgaben der Entwicklung und Durchführung, die dispositiven Aufgaben der Steuerung und Verwaltung sowie die strategische Aufgabe der Prozessverbesserung. Die verschiedenen Aufgaben bei der Entwicklung und Durchführung von Anforderungen, wie die Ermittlung, die Analyse und die Dokumentation, stehen dabei nicht nebeneinander, wie die Abbildung 11.2 vermuten lassen könnte, sondern sind eng miteinander verzahnt. Diese Tätigkeiten bilden den Schwerpunkt dieser Ausarbeitung, wie in Abschnitt 11.1 begründet. Die erste Tätigkeit, die Anforderungsermittlung, umfasst Aktivitäten, die notwendig sind, um die

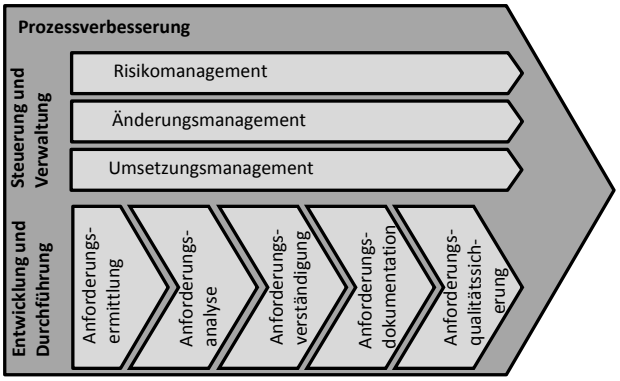


Abb. 11.2 Hauptaufgaben des Requirement Engineerings nach [9, S. 33]

Anforderungen der Kunden zu erfahren. Diese werden bei der Anforderungsanalyse strukturiert, auf Unstimmigkeiten überprüft und verfeinert. Auf verschiedene Methoden, mit denen Anforderungen ermittelt und analysiert werden können, wird in Abschnitt 11.5 genauer eingegangen. Anforderungsverständigung umfasst Aufgaben die dafür geeignet sind, ein gemeinsames Verständnis der Anforderungen herbeizuführen. Dieses gemeinsame Verständnis soll das Projektteam und auch den Kunden umfassen. Auf diese Aufgaben wird in Abschnitt 11.3 näher eingegangen. Die Tätigkeiten der Aufgabendokumentation beschreiben den Vorgang Anforderungen festzuhalten. Darauf wird in Abschnitt 11.4 genauer eingegangen. Um die Qualität der Anforderungen sicherzustellen, werden Tätigkeiten im Rahmen der Anforderungsqualitätssicherung ausgeführt. Dies kann zum Beispiel Reviews der Anforderungen beinhalten. Auf diese Thematik wird kurz in Abschnitt 11.6.1 eingegangen. [9, S. 33] Diese operativen Tätigkeiten finden sich ähnlich bei [8] unter den Teilaspekten Erheben, Dokumentieren und Prüfen.

Die dispositive Tätigkeit der Steuerung und Verwaltung unterteilt [9] in Risikomanagement, Änderungsmanagement und Umsetzungsmanagement. Auf diese Aufgaben wird nur kurz eingegangen, da sie im Kontext der Arbeit, wie in 11.1 dargestellt, nur am Rande auftauchen. Das Umsetzungsmanagement verwaltet und verfolgt die Anforderungen im Laufe des Projekts. Das Änderungsmanagement beschäftigt sich mit der Änderung von Anforderungen, die in jedem Projekt auftauchen und erarbeitet Prozesse um diese Änderungen in das Projekt einzubringen. Das Risikomanagement versucht Risiken in Anforderungen zu identifizieren und abzuschätzen. Bei den Steuerungs- und Verwaltungstätigkeiten ist [9] deutlich präziser als [8]. In [8] werden Tätigkeiten wie Risikomanagement, Änderungsmanagement und Nachverfolgung in einzelnen Kapiteln erläutert, aber nicht bei den zum Requirements Engineerings gehörenden Tätigkeiten erwähnt.

11.3 Funktionen von Anforderungen

Nachdem im vorhergehenden Abschnitt Anforderungen definiert wurden, soll nun aufgezeigt werden, welche Funktionen Anforderungen erfüllen. Dadurch sollte klar werden, warum es wichtig ist, sich mit dem Thema Anforderungen zu beschäftigen. Dies führt darüber hinaus zu einem ersten Verständnis der Anforderungen in ihrem Kontext, dem Projekt.

11.3.1 Anforderungen als Kommunikationsgrundlage

Die Hauptfunktion von Anforderungen ist die Grundlage für Kommunikation. Anforderungen ermöglichen es über das System zu sprechen, eine Vorstellung von dem System zu bekommen und sie bilden auch die Grundlage für Diskussionen über das System, aber auch über die Anforderungen selbst. Damit ermöglichen sie es ein gemeinsames Verständnis innerhalb des Teams und des Kunden zu schaffen. [8, S. 18]

Durch dieses gemeinsame Verständnis ist es möglich, sich auf ein gemeinsames Ziel zu fokussieren und damit der Gefahr entgegen zu wirken, etwas zu entwickeln, was so nicht gewünscht gewesen ist. Nur wenn alle am Projekt Beteiligten ein gemeinsames Verständnis davon haben, was die Vision und die Anforderungen an das Projekt sind, kann jeder einzelne seine Aufgaben sinnvoll und erfolgreich lösen. [6, S. 11]

11.3.2 Anforderungen als rechtliche Grundlage

Die Gesamtheit aller Anforderungen wird zumeist in der Anforderungsspezifikation zusammengefasst. Diese Anforderungsspezifikation wird oft den vertraglichen Vereinbarungen zu Grunde gelegt. Anforderungen bekommen damit eine rechtliche Bindung, so dass das Nichterfüllen von Anforderungen zu Strafen führen kann. Die Abnahme eines Systems erfolgt deshalb auf Grundlage der Anforderungen. [8, S. 18] [3, S. 107]

11.3.3 Anforderungen als Grundlage für Optimierungen

Anforderungen bieten Möglichkeiten für Optimierungen in einem Projekt. Durch die in Abschnitt 11.3.1 erläuterte Funktion als Kommunikationsgrundlage werden Risiken vermieden, was dazu führen sollte, dass das Produkt rechtzeitig und mit dem gewünschten Funktionsumfang fertig gestellt wird. [6, S. 18] Diese Optimierung in Bezug auf die benötigte Zeit und die benötigten Ressourcen lässt sich vor allem

durch die reduzierte Anzahl an Nacharbeiten und Änderungen auf Grund falscher Annahmen zurück führen. [6, S. 14] Fehler lassen sich desto leichter korrigieren, je früher sie auffallen. Am einfachsten ist die Korrektur eines Fehlers bereits in den Anforderungen. [6, S. 29]

11.4 Darstellungsformen von Anforderungen

Bisher wurden Anforderungen definiert und ihre Funktionen erläutert. Dabei wurden kurze natürlich-sprachliche Beispiele für Anforderungen bereits genutzt. Es gibt allerdings vielfältige Möglichkeiten, mit denen Anforderungen dargestellt werden können. In diesem Abschnitt werden ausgewählte Methoden vorgestellt. Die Auswahl beschränkt sich auf solche Darstellungsformen, die in dem in Abschnitt 11.1 gegebenen Kontext verwendbar sind. Die Art der zu wählenden Darstellung hängt von mehreren Faktoren ab. Es muss bedacht werden, welchen Wissensstand die am Projekt beteiligten Personen haben. Darüber hinaus muss auch die Art des zu entwickelnden Systems beachtet werden. Bei sicherheitskritischen komplexen Systemen sind vermutlich andere Notationen zu wählen als bei sicherheitsunkritischen kleineren Systemen. Auch sind in einem sich häufig ändernden Umfeld sicherlich dafür passende Anforderungsnotationen zu wählen. [6, S.96] Nach der Darstellung ausgewählter Formen für Anforderungen wird in Abschnitt 11.4.5 eine Empfehlung für verwendbare Darstellungsformen in dem aufgezeigten Kontext gegeben.

11.4.1 Qualität der Anforderungen

Bevor spezifische Notationen für Anforderungen vorgestellt werden, sollen zunächst einige allgemeine Hinweise festgehalten werden, die auf alle Notationsarten zutreffen. In der Literatur finden sich dafür verschiedene Listen von Eigenschaften, die von Anforderungen erfüllt werden sollten. Diese Listen sind zum Teil mehrere Seiten lang, wie bei [3, S. 127], bis hin zu nur wenige Worte umfassend, wie bei [2, S. 17]. An dieser Stelle wird die Auflistung von [9] präsentiert. Diese besitzt zum einen eine angemessene Länge und stimmt zum anderen in sehr vielen Punkten mit den anderen Aufzählungen überein. Nach [9] sollte eine Anforderung *korrekt* sein, also die richtige Anforderung beschreiben. Sie sollte *vollständig* sein, also die Anforderung vollständig spezifizieren. Anforderungen müssen *eindeutig*, also präzise sein. Sie muss *konsistent*, also widerspruchsfrei sein. Darüber hinaus muss sie *gültig* sein, also mit allen Projektteilnehmern abgestimmt worden sein. Die Anforderung muss eine *Priorität haben*. Es muss dadurch klar sein, wie wichtig sie ist. Sie sollte *verifizierbar*, also testbar sein. Wichtig ist ebenso die *Nachvollziehbarkeit*, also ob der Kontext der Anforderung bekannt ist. Sie sollte zudem *verständlich* und *umsetzbar* sein. [9, S.177]

Notationen von Anforderungen sollten möglichst diese Gütekriterien von Anforderungen unterstützen, um so zu verhindern, dass „schlechte“ Anforderungen aufgenommen werden.

11.4.2 User Stories

Eine User Story beschreibt eine Anforderung aus Sicht des Benutzers. Sie macht deutlich, was ein User benötigt. Ein Beispiel dafür ist „Ein Nutzer kann sein Resümee auf der Webseite posten.“. Wie in diesem Beispiel wird zu Beginn immer deutlich gemacht, welcher Benutzer etwas möchte. Anschließend wird kurz beschrieben, was der Benutzer können möchte. [2, S. 4] Da es bei User Stories darauf ankommt den Benutzern einen bestimmten Mehrwert zu bieten, werden User Stories häufig um einen zweiten Teil ergänzt, der mit „so dass“ beginnt. In dem Beispiel oben könnte das sein „Ein Nutzer kann sein Resümee auf der Webseite posten, so dass er den Betreibern der Internetseite Feedback geben kann.“. Durch diesen Zusatz wird das Verständnis der User Story vertieft und die am Projekt Beteiligten bekommen ein besseres Verständnis der Anforderung. So könnte dieses Resümee mit der Erweiterung oben per E-Mail direkt an den Webseitenbetreiber gehen. Würde die Erweiterung lauten „so dass er sich mit anderen Nutzern über den Inhalt der Webseite austauschen kann“, würde dies eine ganz andere Art der Implementierung benötigen. [4]

User Stories sind, bis auf den Beginn der Formulierung und den Nebensatz, informale Notationsweisen und haben die damit verbundenen Vor- und Nachteile. Sie sind einfach und damit schnell zu lernen. Zudem sind sie bereits vielen Leuten bekannt. Allerdings können sie mehrdeutig, inkonsistent und unvollständig sein. Der Nachteil der informalen Notationsweise wird ein wenig ausgeglichen, indem gefordert wird, dass der Nutzer an den Satzanfang gestellt wird und dass seine Motivation in einem Nebensatz notiert werden muss. Darüber hinaus können User Stories allerdings sehr lückenhaft notiert werden. [6, S.95]

11.4.3 Use-Case Notation

Die Use-Case Notation hat den Vorteil leicht erlernbar zu sein, weil sie lediglich in natürlicher Sprache formuliert wird. Die in Abschnitt 11.4.2 dargestellten Nachteile von informalen Notationsweisen der Unvollständigkeit, Inkonsistenz und Mehrdeutigkeit werden durch eine formularartige Notation ausgeglichen. Dieses Formular kann so angelegt werden, dass ansonsten leicht vergessene Bestandteile von Anforderungen berücksichtigt werden. Zudem ermöglicht das Formular einen schnellen Überblick über benötigte Informationen. [8, S. 202]

Ein mögliches Formular kann auf dem IEEE Standard 830 basieren, der Vorschläge für die Spezifikation von Software Anforderungen gibt.

Titel	*Anforderungsnr.*	*Status*	*Priorität*	*Aufwand*
Essen bestellen	Essen 03 - 06	offen	sehr kritisch	40 PT
Beschreibung	Ein registrierter Gast bestellt Speisen.			
Einschränkungen	Nur Speisen aus vorrätigen Zutaten können bestellt werden.			
Begründung	Der Gast hat Hunger.			
Querbezüge	Küche 02 - 05 Weiterleitungszeit			
Einflüsse	Mobilteil, Küchendisplay			
Akzeptanzkriterien	Testszenarios Essen bestellen 03 und Essen bestellen 04			
Kommentare	Dieser Use-Case soll im Rahmen der Hannover Messe am 12.10.2014 vorgestellt werden.			

Tabelle 11.1 Formular für Use Cases nach [3, S. 116] und [8, S. 202]

Wie ein solches Formular aussehen kann, ist in Tabelle 11.1 zu sehen. Der *Titel* sowie die *Anforderungsnummer* erhöhen die Verfolgbarkeit der Anforderung. Es ist hier auf eine sinnvolle Vergabe der Anforderungsnummer zu achten. Der *Status* ist wichtig, um nachvollziehen zu können, ob alle Anforderungen tatsächlich erfüllt wurden. In der *Beschreibung* soll kurz und präzise die Anforderung beschrieben werden. Sie kann auf weitere Dokumente verweisen. *Einschränkungen* kann auf nichtfunktionale Anforderungen verweisen, die diese Anforderung einschränken. Ebenso können auf Gesetze verwiesen werden. Das Feld *Begründung* soll den Nutzen der Anforderung erläutern. Dies kann bei der Umsetzung der Anforderung helfen. Die *Priorität* legt fest, wie wichtig die Aufgabe für das Projekt ist. Bei *Querbezüge* können Abhängigkeiten und Erweiterungen aufgezeigt werden. In *Aufwand* wird geschätzt, wie viel Aufwand der Use - Case benötigt. Dies ist wichtig als Planungsgrundlage. Wichtig für die Abnahme der Anforderung sind die *Akzeptanzkriterien*. Hier können Testfälle vorgeben werden. Im *Kommentarfeld* können weitere Hinweise hinterlegt werden, die im Laufe des Lebenslauf des Use-Cases entstehen. [3, S. 116]

11.4.4 UML Diagramme

Eine formalere Notation für Anforderungen sind Unified Modeling Language (UML) Diagramme. Diese Sprache hat die Funktion Systeme zu modellieren. Für diesen Zweck gibt es verschiedene Diagrammtypen. [1, 191]

Interessant für die Modellierung von Anwendungsfällen ist das UML Use - Case Diagramm, das auch die Grundlage für die Anforderungserhebungsmethode Anwendungsfälle ist, wie in Abschnitt 11.5.4 beschrieben. In Use - Case Diagrammen werden Akteure sowie Use - Cases aufgenommen. Die Use - Cases werden mit den Akteuren verbunden, um die an einem Use - Case beteiligten Akteure aufzuzeigen. Use - Case Diagramme bieten damit die Möglichkeit einen ersten Überblick über ein System mit seinen Akteuren und den im System ausgeführten Use - Cases zu bekommen. [10, S. 124]

Die beiden nächsten vorgestellten UML Diagrammtypen ergeben sich aus dem in Abschnitt 11.1 dargestellten Kontext. Die Verwendung des Sequenzdiagramms ist sinnvoll, weil im dargestellten Projekt wahrscheinlich verschiedene Systeme miteinander kommunizieren. Darüber hinaus ist die Verwendung des Aktivitätsdiagramms angemessen, da wahrscheinlich einige Verarbeitungsschritte und -abläufe dargestellt werden müssen.

UML Aktivitätsdiagramme visualisieren insbesondere Aktivitäten, die zusammen genommen einen Systemvorgang ergeben. Sie visualisieren unter anderem den Kontrollfluss während eines Vorgangs, indem Aktivitäten durch Pfeile miteinander verbunden werden, die eine Weitergabe der Kontrolle darstellen. [10, S. 123] Im Zusammenhang mit Anforderungen sind Aktivitätsdiagramme dazu geeignet einen Anwendungsfall zu visualisieren oder aber eine Teiloperation eines Use - Cases. Dabei sind Aktivitätsdiagramme besonders dann vorteilhaft, wenn Entscheidungen oder Alternativen in einem Anwendungsfall berücksichtigt werden müssen. [8, S. 205]

Das UML Sequenzdiagramm wird zur detaillierten Visualisierung von Kommunikation verwendet. In einem Sequenzdiagramm werden die Akteure sowie ihre Aktivitäten visualisiert. Dabei wird besonders Wert auf die Interaktion und die Art der Interaktion der Akteure miteinander gelegt. [10, S. 126] Im Kontext der Anforderungsbeschreibung werden Sequenzdiagramme verwendet, wenn die ausgetauschten Nachrichten zwischen Akteuren visualisiert werden sollen. Da Sequenzdiagramme sehr detailliert sind, werden sie meist erst zur genaueren Anforderungsspezifikation verwendet. [8, S. 213]

11.4.5 Empfehlung

Ein Vorteil des Kontextes dieser Arbeit ist, dass alle beteiligten Personen technisch versiert sind, wie in Abschnitt 11.1 dargelegt. Dies hat den Vorteil, dass hier auch im Allgemeinen schwerer verständliche Diagramme, wie zum Beispiel das UML Sequenzdiagramm verwendet werden können. Zum Einstieg des Projekts ist es empfehlenswert UML Use - Case Diagramme zu verwenden, weil die am Projekt Beteiligten die Methodenkenntnisse dafür mitbringen, Use - Case Diagramme einen guten Überblick bieten, sie bei der Erhebungsmethode Anwendungsfälle verwendet werden, wie in Abschnitt 11.4.4 erläutert und darüber hinaus im Rational Unified Process verwendet werden. [8, S. 220]

Für die genauere Notation sollte die Use-Case Notation verwendet werden. Diese eignen sich ebenfalls wie das UML Use - Case Diagramm für eine Einführung der Anforderung besonders und sind zudem sehr gut mit dem UML Use - Case Diagramm kombinierbar. Wie in Abschnitt 11.4.3 erläutert, sind sie einfach erlernbar und unterstützen dennoch bei der Formulierung vollständiger Anforderungen. Im weiteren Verlauf des Projekts sollten dann vertiefend dazu weitere UML Diagramme verwendet werden, um die Anforderungen noch genauer zu spezifizieren. Nach

dem in Abschnitt 11.2.1 gegebenen Querschnittsmodell sollten die weiteren UML Diagramme ab den Systemdesignanforderungen verwendet werden. [8, S.220]

Neben den Notationen selbst ist es sehr wichtig die Verknüpfungen zwischen den verschiedenen Darstellungsformen zu behalten. Dadurch kann sichergestellt werden, dass der Kontext einer Implementierung durch die verschiedenen Schichten der Anforderungsverfeinerung hindurch bis hin zu einer Kundenanforderung nachvollzogen werden kann. Dafür muss eine geeignete Organisationsform gefunden werden. [9, S. 162]

11.5 Methoden zur Anforderungserhebung

In diesem Abschnitt werden verschiedene Methoden zur Anforderungserhebung vorgestellt. Dabei wurden Methoden ausgewählt, die in dem Abschnitt 11.1 vorgestellten Kontext praktikabel sind. Ein Beispiel für eine nur eingeschränkt praktikable Methode ist die Marktstudie. [1, S. 40] Dafür stehen weder die Kenntnisse noch die Mittel im Rahmen des Projekts zur Verfügung.

Stattdessen wurden solche Methoden gewählt, die sich für die Sammlung von Anforderungen eignen, die in einem Dialog miteinander stattfinden und die im gegebenem Kontext anwendbar sind. Diese Methoden sind das Interview in Abschnitt 11.5.2 und der Workshop in Abschnitt 11.5.3. In den beiden darauf folgenden Abschnitten Abschnitt 11.5.4 und 11.5.5 werden mit diesen ersten beiden Methoden kombinierbare Methoden vorgestellt. [9, S.198]

Bevor aber Methoden vorgestellt werden, wird in Abschnitt 11.5.1 zunächst allgemein auf Probleme bei der Anforderungserhebung eingegangen.

11.5.1 Probleme bei der Anforderungserhebung

Bei der Erhebung von Anforderungen können verschiedene Probleme auftreten. So können zu Beginn bei verschiedenen Projektteilnehmern unterschiedliche Erwartungen und Vorstellungen von dem System vorhanden sein. Dies kann in späteren Stadien des Projekts zu erheblichen Problemen führen, weil Anforderungen durch missverstandene Kontextvorstellungen anders interpretiert werden. [8, S. 25]

Ein weiteres häufiges Problem sind mangelnde Qualität in den Anforderungen. So kann es passieren, dass Anforderungen mehrdeutig sind, dass redundante oder widersprüchliche Anforderungen vorhanden sind oder zu ungenaue Angaben gegeben worden sind. Einige dieser Probleme wurden bereits bei den Darstellungsformen und der Empfehlung zur Nutzung in Abschnitt 11.4.5 adressiert. Darüber hinaus verdeutlicht dieses Problem auch die Wichtigkeit der Zusammenarbeit mit dem Qualitätsmanagement. Während der Anforderungserhebung ist bereits darauf zu achten, dass die Anforderungen auf diese Qualitätskriterien überprüft werden. [9, S. 274]

11.5.2 Interview

Beim Interview werden Fragen an den Kunden vorbereitet, die anschließend in einem Gespräch geklärt werden. Diese Methode eignet sich besonders für die initiale Anforderungsermittlung, um einen ersten Eindruck zu bekommen. [8, S. 124] Allerdings benötigt ein Interview einiges an Vorbereitung. Um ein Interview machen zu können, müssen sich die Interviewer zunächst in das Thema einarbeiten, um einen Fragenkatalog erstellen zu können. Anschließend wird ein Fragenkatalog, sowie Templates zur Erfassung der Antworten vorbereitet. [1, S. 35] Nachdem das Interview durchgeführt wurde, werden die Antworten auf Vollständigkeit und Plausibilität geprüft, um anschließend protokolliert und analysiert zu werden. [9, S. 203]

11.5.3 Workshop

Der Workshop eignet sich ebenso wie das Interview für eine erste Anforderungsermittlung. Er benötigt mehr organisatorische Vorbereitung als ein Interview, weil eine Agenda erarbeitet werden muss, sowie Raum und Materialien vorbereiten werden müssen. [9, S. 206] Im Gegensatz zum Interview wird aber keine so tiefe Einarbeitung in fachliche Themen benötigt, weil die Teilnehmer des Workshops zusammen die Anforderungen erarbeiten. Er eignet sich somit besonders dann als Erhebungsmethode, wenn ein Informationsungleichgewicht zwischen den Teilnehmern vorhanden ist. [8, S. 128]

11.5.4 Anwendungsfälle

Anwendungsfälle ist eine Erhebungstechnik, die zur Definition der erwarteten Leistungen eines System genutzt werden kann. Hierfür wird das in Abschnitt 11.4.4 vorgestellte UML Use - Case Diagramm verwendet, um Akteure und ihre Interaktionen in dem System abzubilden. Dazu werden zunächst die Akteure identifiziert. Akteure sind Nutzer des Systems, aber auch andere Systeme, mit denen das System interagieren soll. Interessant ist die Frage danach, wer Informationen in das System eingibt und wer Informationen entnimmt. Nach der Identifizierung von Akteuren werden Anwendungsfälle aufgestellt. Durch die Anwendungsfälle wird notiert, was die verschiedenen Akteure von dem System erwarten und wie sie mit dem System interagieren werden. Sobald diese Anwendungsfälle aufgestellt worden sind, wird daraus ein Use - Case Diagramm erstellt, um einen Überblick über das System mit allen Akteuren und ihren Interaktionen zu bekommen. Abschließend werden die Anwendungsfälle genauer, zum Beispiel mit einer Use - Case Notation, spezifiziert. [9, S. 223]

Bei der Erhebungstechnik Anwendungsfälle ist zu beachten, dass sie für die initiale Erhebung geeignet ist. Die Anwendungsfälle werden dabei noch nicht vollständig

spezifiziert, so werden unter anderem noch keine Aussagen zu einem zeitlichen Verlauf getroffen.[8, S. 184] Wie in Abschnitt 11.4.5 erläutert, muss bei einer vollständigen Spezifikation auf Sequenzdiagramme, Aktivitätsdiagramme oder andere Darstellungsformen zurückgegriffen werden.

11.5.5 Kreativitätstechniken

Insbesondere bei der ersten Anforderungserhebung ist es wichtig der Kreativität freien Lauf zu lassen und Begeisterung zu schaffen. Somit bieten sich als eine Unterstützungstechnik für die Anforderungserhebung Kreativitätstechniken an.

Bei der Kreativitätstechnik Brainstorming werden zunächst Ideen gesammelt und diese nicht weiter bewertet. Dadurch kann eine größere Bandbreite von Anforderungen aufgedeckt werden, die allerdings anschließend analysiert und gefiltert werden müssen. [8, S. 116] Eine weitere Kreativitätstechnik für die erste Ermittlung von Anforderung ist die Mindmap. Ähnlich wie bei einem Brainstorming werden hier erste Ideen gesammelt, allerdings werden diese direkt strukturiert. [9, S. 196]

11.5.6 Empfehlung

Für eine erste Erfassung von Anforderungen in dem hier gegebenen Kontext kommen das Interview und der Workshop in Frage. Problematisch beim Interview sind die vorzubereitenden Fragen. Das Wissen ist in diesem Projekt sehr ungleich verteilt, was eine Vorbereitung des Interviews sehr erschwert. Hierin liegt gleichzeitig die Stärke des Workshops. In einem Workshop können die verschiedenen Fähigkeiten der Teilnehmer genutzt werden, um einen gemeinsamen Kenntnisstand zu erarbeiten.

Ein weiterer Vorteil des Workshops ist auch, dass er das in Abschnitt 11.5.1 beschriebene Problem der unterschiedlichen Erwartungen und Zielvorstellungen durch eine Kommunikation mit allen Projektteilnehmern reduziert. Zudem lässt er sich mit verschiedenen anderen Methoden kombinieren, so auch mit der Methode Anwendungsfälle. Anwendungsfälle wiederum scheinen eine sehr geeignete Methode für ein Projekt, in dem als Prozessmodell RUP verwendet wird.

Darüber hinaus lassen sich in einem Workshop direkt Probleme mit der Qualität adressieren. Es können Phasen eingeplant werden, bei denen die Anforderungen zum Beispiel auf Redundanz und Widersprüchlichkeit überprüft werden.

Interviews sind aber durchaus geeignet, um später auftauchende Fragestellungen zu klären. Zu dem Zeitpunkt sollten schon sehr ähnliche Zielvorstellungen zwischen den Projektteilnehmern vorhanden sein und auch das notwendig Know-How, um die Fragen zu formulieren. Zudem wäre es für kleine spezielle Fragestellungen zu aufwendig, wenn dafür alle Projektteilnehmer versammelt werden.

Die Empfehlung ist somit ein Workshop zur ersten Ermittlung der Anforderungen und zur genaueren Spezifizierung der Anforderungen Interviews.

11.6 Anforderungen im Projektalltag

Wie in [6, S. 39] erläutert, ist das Anforderungsmanagement eng mit anderen Managementaufgaben in einem Projekt verknüpft. Deshalb sollen in diesem Abschnitt die mit dem Requirementsengineering zusammenhängenden Disziplinen Qualitätsmanagement in Abschnitt 11.6.1 und Projektmanagement in Abschnitt 11.6.2 dargestellt werden.

11.6.1 Anforderungen im Kontext Qualitätsmanagement

In 11.4.1 wurden verschiedene Qualitätsmerkmale aufgezeigt, denen Anforderungen entsprechen sollten. Damit die Anforderungen diesen Merkmalen entsprechen, ist es notwendig, dass entsprechende Qualität sichernde Maßnahmen ergriffen werden. Ein wichtiges Verfahren dafür ist das Review von Anforderungen. Dafür muss zunächst einmal das Ziel der Prüfung festgelegt werden. Sollen formale Aspekte, inhaltliche und fachliche Aspekte oder Weiterverwendbarkeit überprüft werden? Daraus wiederum ergeben sich erste Anhaltspunkte zur Auswahl der Prüfer. Es muss festgelegt werden, wer und wie viele die Anforderungen prüfen sollen. Danach können verschiedene Prüftechniken angewendet werden. Mögliche Verfahren sind die Stellungnahme durch einen Experten, eine Inspektion durch am Projekt beteiligte Personen oder das Erstellen eines Prototypen. Es ist wichtig, dass solche Maßnahmen im Projekt eingeleitet werden, allerdings würde es den Rahmen dieser Arbeit überschreiten, detailliert auf diese einzugehen. [8, S. 306]

11.6.2 Anforderungen im Kontext Projektmanagement

Das Projektmanagement hat Einfluss auf den Prozess des Anforderungsmanagement. So gibt es in [6, S. 39] eine Zuordnung von Prozessmodellen fürs Projektmanagement zu Prozessmodellen für das Requirements Engineering. Auch andere Quellen verweisen auf Prozessmodelle, belassen es allerdings bei einer kurzen Schilderung und verweisen auf die Phasen, in denen Requriements Engineering stattfindet. Im Rational Unified Process (RUP) sind Anforderungen eine Disziplin. Diese Disziplin hat die Aufgabe das Problemumfeld zu analysieren, die Bedürfnisse der Stackholder zu verstehen, das System sowie den Systemumfang zu definieren, Veränderungen bei Anforderungen zu verwalten und Systemdefinitionen zu verfeinern. Dafür gibt RUP ein Aktivitätsdiagramm vor, bei dem aber die Erhebung der Anforderungen nicht explizit aufgezählt wird. Dies ist im RUP ein Unterpunkt der Problemanalyse. Somit trifft der RUP keine Vorgaben darüber in welcher Form Anforderungen ermittelt werden sollen. Allerdings ist der RUP besonders Dokumenten und insbesondere UML Diagramm getrieben. Somit bieten sich solche Verfahren an, die UML Diagramme als Output produzieren. [8, S. 48] [9, S. 88] [3, S. 47] [1, S. 13]

Wichtig beim Verhältnis vom Requirements Enginnering zum Projektmanagement ist die direkte Zusammenarbeit und das aufeinander angewiesen sein. Nur mit gut spezifizierten Anforderungen, die passende Schätzungen aufweisen, ist es möglich ein Projekt zu planen. Umgekehrt ist das Requirements Engineering darauf angewiesen, dass es Zeit und Ressourcen vom Projektmanagement zugewiesen bekommt, um seine Arbeit angemessen ausführen zu können. [5, S. 11]

11.7 Fazit und Ausblick

Auf Grund der Kürze der Ausarbeitung kann diese nicht mehr als eine Einführung in das Anforderungsmanagement darstellen. In der Literatur finden sich ganze Bücher zu Themen, die in dieser Ausarbeitung in nur einem Abschnitt dargestellt werden, wie zum Beispiel Anforderungsmanagement im Kontext Projektmanagement in [5] oder User Stories in [2]. Auch wurden einige Themen ausgeklammert. So wurde das Thema Reviews von Anforderungen nur sehr kurz vorgestellt, weil in dieser Ausarbeitung die Grundlagen für ein Requirements Engineering erläutert werden sollen. Auch wurde das Thema des Anforderungsmanagements lediglich angeschnitten.

Diese Grundlagen sollen dabei helfen ein erstes Requirements Engineering aufzubauen. Das Thema Requirements Engineering ist zu komplex, um beim Initiieren direkt alle Teilaspekte mit einführen zu können. Zum einen haben die Projektteilnehmer kaum Kenntnisse über das Requirements Engineering und zum anderen ist das Projekt noch in der Anfangsphase. Die verschiedenen Aspekte müssen aber im Hinterkopf behalten werden, um sie in späteren Projektphasen aufzugreifen und zu etablieren.

Genauso wichtig ist es für den Verlauf des Projekts, dass qualitätsfördernde Maßnahmen umgesetzt werden, wie sie in Abschnitt 11.6.1 vorgestellt wurden. Nur so kann sichergestellt werden, dass Anforderungen ihre Funktionen erfüllen. Aber auch diese müssen nach und nach aufgegriffen und etabliert werden, um nicht zu einer Überforderungen zu führen.

Eine weitere Aufgabe im Laufe des Projekts wird es sein, eine nachvollziehbare Organisationsform für Anforderungen zu finden. Wie in Abschnitt 11.4.5 dargestellt, ist es notwendig, dass Anforderungen von der Kundenanforderungen bis hin zur Implementierung nachvollziehbar bleiben.

Für eine weitere wissenschaftliche Auseinandersetzung eignet sich die Tatsache, dass in der Literatur keine einheitliche Systematik für Anforderungsarten gegeben wird, sondern im Gegenteil sich widersprechende Systematiken verwendet werden, wie in Abschnitt 11.2.1 dargestellt. Genauso sinnvoll ist eine Beschäftigung mit der Definition von Requirements Engineering, die durchaus sehr unterschiedlich ist, wie in Abschnitt 11.2.2 gezeigt.

Literaturverzeichnis

1. Chemuturi, M.: Requirements Engineering and Management for Software Development Projects, 1 edn. Springer Verlag (2013)
2. Cohn, M.: User Stories Applied, 17 edn. Addison-Wesley (2012)
3. Ebert, C.: Systematisches Requirements Management. Anforderungen ermitteln, spezifizieren, analysieren und verfolgen, 1 edn. dpunkt.verlag (2005)
4. Eocha, C.O., Conboy, K.: The role of the user story agile practice in innovation. Lean enterprise software and systems pp. 20–30 (2010)
5. Fahney, R., Gartung, T., Glunde, J., Hermann, A., Hoffmannn, A., Knauss, E., Valentini, U., Weißbach, R.: Requirements Engineering und Projektmanagement, 1 edn. Springer Verlag (2013)
6. Hood, C., Wiebel, R.: Optimieren von Requirements Management & Engineering, 1 edn. Springer Verlag (2005)
7. Hull, E., Jackson, K., Dick, J.: Requirements Engineering, 3 edn. Springer Verlag (2011)
8. Rupp, C., die SOPHISTen: Requirements-Engineering und -Management. Professionelle, iterative Anforderungsanalyse für die Praxis, 4 edn. Carl Hanser Verlag (2007)
9. Schienmann, B.: Kontinuierliches Anforderungsmanagement. Prozesse - Techniken - Werkzeuge, 1 edn. Addison-Wesley (2002)
10. Sommerville, I.: Software Engineering, 9 edn. Pearson Education (2011)
11. Sommerville, I., Sawyer, P.: Requirements Engineering. A good practice guide, 7 edn. John Wiley & Sons (2006)

A.5 C++ Stil Richtlinien

Im Folgenden werden C++ Stil Richtlinien für das Projekt beschrieben. Die C++ Richtlinien helfen dabei den Code lesbarer zu gestalten und erhöhren dadurch die Wartbarkeit eines Softwaresystems. Die folgenden Richtlinien orientieren sich an dem Google C++ Style Guide ([18]).

A.5.1 Sprache

Im Quellcode wird britisches Englisch verwendet. Dies bedeutet, dass sowohl die Namensgebung der Variablen, Klassen und weiteren Objekte auf Englisch erfolgt, wie auch die Kommentare auf Englisch geschrieben werden. Durch die Verwendung von Englisch im Code wird dieser einem großen Benutzerkreis zugänglich.

A.5.2 Namensgebung

Die Namensgebung ist der wichtigste Bestandteil der C++ Stil Richtlinien, um Konsistenz sicherzustellen. Durch eine konsistente Namensgebung wird sichergestellt, dass der Leser durch den Namen eines Objekt bereits weiß, worum es sich bei dem Objekt handelt.

A.5.2.1 Generelle Namensregeln

Der Name muss sinnvoll sein, was in diesem Fall beschreibend bzw. sprechend meint. Es sollte also durch den Namen bereits klar sein, was sich hinter der Variable verbirgt und wofür sie verwendet wird. Abkürzungen sollten nur verwendet werden, wenn sie allgemein bekannt sind. Ausnahmen von dieser Regel sind i oder j als Iterator. Die Begriffe im Namen müssen aus dem Englischen stammen, wie oben beschrieben. Im Folgenden finden sich zunächst positive und danach negative an [18] angelehnte Beispiele.

```
int price_count_reader; // No abbreviation.
int num_errors; // "num" is a widespread
    convention.
int num_dns_connections; // Most people know what
    "DNS" stands for.
```

```
int n; // Meaningless.
int nerr; // Ambiguous abbreviation.
int n_comp_conns; // Ambiguous abbreviation.
```

```
    int wgc_connections; // Only your group knows
        what this stands for.
    int pc_reader; // Lots of things can be
        abbreviated "pc".
    int cstmr_id; // Deletes internal letters.
```

A.5.2.2 Dateinamen

Dateinamen sollten nur kleine Buchstaben beinhalten und als Separator „_“. C++ Dateien sollten als Endung .cc haben und Header Dateien die Endung .h. Die Dateinamen sollten möglichst ausführlich sein, damit es keine Überlappungen mit anderen Namen geben kann. Zudem sollten die Dateien möglichst mit dem Klassennamen übereinstimmen. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
my_useful_class.cc
my-useful-class.cc
myusefulclass.cc
myusefulclass_test.cc
```

A.5.2.3 Typennamen

Alle Typen, wie Klassen, Strukturen, Typendefinition und Enums sollten mit einem Großbuchstaben anfangen und in Camelcase geschrieben werden, also jedes neue Wort im Namen fängt wieder mit einem Großbuchstaben an. Neben dem Großbuchstaben wird kein Separator verwendet. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
// classes and structs
class UrlTable { ...
class UrlTableTester { ...
struct UrlTableProperties { ...

// typedefs
typedef hash_map<UrlTableProperties *, string>
    PropertiesMap;

// enums
enum UrlTableErrors { ...
```

A.5.2.4 Variablennamen

Variablen sollten nur kleine Buchstaben und als Seperator „_“ beinhalten. Im Folgenden finden sich zunächst positive und danach negative an [18] angelehnte Beispiele.

```
string table_name;  // OK - uses underscore.
string tablename;   // OK - all lowercase.

string tableName;   // Bad - mixed case.
```

Klassenvariablen

Klassenvariablen werden wie normale Variablen bezeichnet, allerdings mit einem abschließenden „_“. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
class TableInfo {
  ...
 private:
  string table_name_;  // OK - underscore at end.
  string tablename_;   // OK.
  static Pool<TableInfo>* pool_;  // OK.
};
```

Structurvariablen

Structurvariablen werden wie normale Variablen bezeichnet. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
struct UrlTableProperties {
  string name;
  int num_entries;
  static Pool<UrlTableProperties>* pool;
};
```

Globale Variablen

Globale Variablen sollten möglichst vermieden werden. Falls sie doch benutzt werden, so sollte ein führendess "g_ßur Benennung einer normalen Variable hinzugefügt werden.

A.5.3 Funktionsnamen

A.5.3.1 Reguläre Funktionen

Reguläre Funktionen werden im Camelcase beschrieben. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
AddTableEntry()
DeleteUrl()
TryToOpenFile()
};
```

A.5.3.2 Accessoren und Mutatoren

Accessoren werden so wie Variablen nur klein und mit einem „_“ als Seperator bezeichnet. Mutatoren werden ebenso, aber mit einem führenden get_ bezeichnet. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
class MyClass {
 public:
  ...
  int num_entries() const { return num_entries_; }
  void set_num_entries(int num_entries) {
       num_entries_ = num_entries; }

 private:
  int num_entries_;
};
```

A.5.4 Namespaces

Namenspaces werden in Kleinbuchstaben bezeichnet. Sie sollten den Projektnamen und Ordnerstrukturen entsprechen.

A.5.5 Enumnamen

Enums sollten wie Konstanten nur mit Großbuchstaben und „_“ als Separator bezeichnet werden. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
enum UrlTableErrors {
```

```
  kOK = 0,
  kErrorOutOfMemory,
  kErrorMalformedInput,
};
enum AlternateUrlTableErrors {
  OK = 0,
  OUT_OF_MEMORY = 1,
  MALFORMED_INPUT = 2,
};
```

A.5.6 Formattierungen

A.5.6.1 Abstandhalter

Als Abstandhalter werden zwei Leerzeichen verwendet. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
class MyClass {
 public:
  ...
  int num_entries() const { return num\_entries\_; }
  void set_num_entries(int num_entries) {
       num_entries_ = num_entries; }

 private:
  int num\_entries\_;
};
```

A.5.6.2 Zeilenlänge

Die maximale Zeilenlänge beträgt 80 Zeichen.

A.5.7 Dateien

A.5.7.1 Header Files

In der Regel sollte jede .cc Datei eine dazugehörige .h Datei besitzen.

A.5.8 Error Handling

Wo möglich werden Exceptions verwendet.

A.5.9 Variable

A.5.9.1 Lokale Variable

Lokale Variablen sollten möglichst nahe zu der ersten Verwendung deklariert werden. Zudem sollten sie direkt deklariert und initialisiert werden. Im Folgenden finden sich zunächst positive und danach negative an [18] angelehnte Beispiele.

```
int j = g();  // Good -- declaration has
    initialization.
```

```
vector<int> v = {1, 2};  // Good -- v starts
    initialized.
```

```
int i;
i = f();      // Bad -- initialization separate from
    declaration.
```

Variablen sollten möglichst in dem kleinst möglichen Scoope verwendet werden. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
while (const char* p = strchr(str, '/')) str = p + 1;
```

Allerdings muss dabei beachtet werden, dass Objekte wenn möglich vor der Schleife deklariert werden und in diesem Fall die Initialisierung getrennt von der Deklaration stattfinden. Im Folgenden finden sich zunächst positive und danach negative an [18] angelehnte Beispiele.

```
Foo f;  // My ctor and dtor get called once each.
for (int i = 0; i < 1000000; ++i) {
  f.DoSomething(i);
}
```

```
// Inefficient implementation:
for (int i = 0; i < 1000000; ++i) {
  Foo f;  // My ctor and dtor get called 1000000
      times each.
  f.DoSomething(i);
}
```

A.5.9.2 statische und globale Variablen

Das Nutzen von Klassen in statischen oder globalen Variablen ist untersagt, weil es schwer zu reproduzierende Fehler nach sich zieht.

A.5.10 Klassen

A.5.10.1 Konstruktoren

Konstruktoren sollten möglichst einfach gehalten sein. Wenn eine komplizierte Initialisierung notwendig ist, sollte sie entweder in Fabriken oder in Initialisierungsmethoden ausgelagert werden. Konstruktoren sollten insbesondere nur in Fehlerfällen Exceptions schmeißen.

Die Konstruktoren sollten mit dem Schlüsselwort explicit bezeichnet werden, wenn sie nur ein Argument oder ein Argument und weitere optionale Argumente hat. Dadurch wird verhindert, dass aus versehen implizite Umwandlungen vorgenommen werden.

A.5.10.2 Initialisierung

Wenn in einer Klasse Membervariablen verwendet werden, so sollten entweder inclass Initialisierer verwendet werden oder mindestens ein Konstruktor, der die Variablen initialisiert.

A.5.10.3 Zugriffe

Memberveriablen sollten private sein und der Zugriff über Accessoren und Mutatoren erfolgen.

A.5.11 Kommentare

Kommentare können mit // oder /* */ gekennzeichnet werden. Es sollte aber innerhalb der Module konsistent erfolgen.

A.5.11.1 Klassen

Für jede Klasse, die exportiert wird, muss dokumentiert werden, was ist Verwendungszweck ist und wie sie verwendet werden kann. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
/** @brief Dummy class used for illustration
    purposes. Doing something with it.

    Detailed description follows here.
    @author X. XYZ, DESY
    @date March 2008
    */
class myClass{
```

Darüber hinaus muss für jede exportierte Methode dokumentiert werden, welche Aufgabe sie hat unter dem Tag @brief. Dazu müssen die Parameter der Methode dokumentiert unter @param dokumentiert werden. Im Folgenden finden sich positive an [18] angelehnte Beispiele.

```
  /**
  * @brief returns a list of data of components saved
      in the time interval from starttime till endtime
  *
  * @param starttime (time_t) start of the interval
  * @param endtime (time_t) end of the interval
  * @return std::list<ComponentData> list of
      componentData tuple
  virtual std::list<ComponentData>
      getComponentData(time_t starttime, time_t
      endtime)=0;
  */
```