TECHNISCHE UNIVERSITÄT BERLIN

Fachgebiet Energieversorgungsnetz und Integration Erneuerbarer Energien

STUDIENARBEIT

Objektorientierte Implementierung und Simulation einer Kältelast mit Kältespeicher im Energieversorgungsnetz

vorgelegt von Juri STEBLAU
Matr.-Nr: 300244

11. August 2011

Korrektoren:

Advisor:	Dipl.-Ing. Felix KLEIN	-	(TU-Berlin)
Examinator:	Prof.Dr.-Ing. Kai STRUNZ	-	(TU-Berlin)

Realisation und Simulation einer Kältelast mit Kältespeicher im Energieversorgungsnetz

Abstract: Hier bitte am Ende der Arbeit auf english die kurze Inhaltsangabe schreiben.

Hier muss noch ein Abstract rein !!!AM ENDE!!!

Keywords: Möglicherweise in Englisch! Fuck!

Todo list

Inhaltsverzeichnis

Abbildungsverzeichnis

MATLAB® -Code Verzeichnis

Abkürzungsverzeichnes

NK_KR_V	normalgekühltes Kühltregal an Verbundanlage
NK_KT_S	normalgekühlte Kühltruhe steckerfertig
OOP	objektorientierte Programmierung
TK_TKT_S	tiefgekühlte Tiefkühltruhe steckerfertig

Kapitel 1

Einleitung

Einleitung mehrfach durchlesen!

DieErforschung der Ursachen und der Folgen des Klimawandels, die wachsende Schwierigkeit bei der Bereitstellung der konventionellen Energien, die Neubewertung der Risiken und technischen Mitteln bei der Endlagerung von Abfällen der Atomindustrie, die besorgniserregende Erkenntnis der bisherigen Fehlbewertung der Atomsicherheit werden gewiss die politisch beschlossene Förderung der erneuerbaren Energien zu einer grundlegenden Energieform in den kommenden Jahren in Deutschland und in Europa forcieren.

Reicht so? Guck bitte nach oben.

Der Umstieg auf alternative Energien ist mit einigen Problemen verbunden. An einigen Orten ist der Einsatz dieser Technik aus politischer, technischer, ökonomischer oder ökologischer Sicht nicht möglich. Darum weichen oft die Stromerzeugung und der Strombedarf zeitlich und räumlich voneinander ab. Windkraft im Meer, Wasserkraft in den Bergen, Sonnenkraft in dem Süden, Geothermie in Island. Desweiteren kann nur ein Teil der erneuerbaren Energien direkt vom Menschen beeinflusst werden. Besonders die Menge der durch Sonne und Wind gewonnenen Energie schwankt abhängig von der Wetterlage. Die Integration dieser Energie in das Netz führt zur erhöhten Bereitstellung an Regelenergie. Diese Herausforderung Hauptsächlich wird durch übermäßige Belastung der zur Ausregelung geeigneten konventionellen thermischen Kraftwerke gelöst. Durch den regelungsbedingten ineffizienten Teillastbetrieb und wiederholte An- und Abfahrvorgänge sinkt der Wirkungsgrad und ein höherer Verschleiß der ist die Folge. Aus langfristiger Sicht wird jedoch

Bewertung einfügen.

Das am Ende zur Ende schreiben

eine Investition in Lastmanagement und in Energiespeicher unentbehrlich sein. Effiziente Lastmanagemant und Energiespeicherkonzepte können helfen diesen Problemen langfristig entgegenzuwirken. Brückentechnologien nicht sicher. Politischer Druck wächst.

Kapitel 2

Theoretische Problembetrachtung

Inhaltsangabe

2.1 Erneuerbare Energien im Energieversorgungsnetz

Die erneuerbaren Energien übernehmen in Deutschland mit jedem Jahr einen größeren Anteil an der Elektrizitätsversorgung. Mehr als die Hälfte der erneuerbaren Energie wird durch Wind und Photovoltaik erzeugt. Diese Energiequellen sind jedoch in hohem Grad witterungsabhängig und können stark fluktuieren. Dadurch kann die Menge der zur Verfügung stehenden Energie von der Nachfrage zeitlich enorm abweichen. Dies kann den sicheren Betrieb des Stromnetzes bei einer konstanten Frequenz von 50 Hz gefärden. Zusätzliche Bereitstellung an Regelleistung wird dadurch notwendig. Außerdem erfolgt die Erstellung der Fahrpläne für den Einsatz der konventionellen Kraftwerke gezwungenermaßen auf den fehlerbehafteten Vortagsprognosen über die Menge an Leistung aus erneuerbaren Energien. Die Güte der Prognose bestimmt dabei direkt den Regel- und den Reservebedarf.

wo ist die Quelle?

Eine exakte Prognose über die Leistung aus Wind- und Photovoltaikanlagen kann nur auf Grundlage exakter Wettervorhersagen erfolgen. In der Literatur wird eine mittlere Genauigkeit für Day-Ahead Windleistungsprognosen von 7 % [17] und für Leistungsprognose aus Photovoltaikanlagen im Bereich von $3,5\,\% - 4,4\,\%$ [8] angegeben. Darüberhinaus können im Verlauf eines Tages zeitweise deutliche Abweichungen von diesen Werten auftreten. Durch den gedrosselten Betrieb der Anlagen ist eine Möglichkeit zur Kompensation der Prognosefehler gegeben. Der gedrosselte Anteil kann dadurch jedoch nicht genutzt werden. Die Bereitstellung der Regelleistung durch konventionelle oder andere Kraftwerke erscheint hier als sinnvoll. Eine Alternative zu den obengenannten Methoden bieten Energiespeicher, da sie bei Überangebot Energie speichern können und bei Unterversorgung Energie in das Netz einspeisen können.

Variable Gestalltung der Netzlast kann weiterhin zur Reduzierung der obengenannten Problematik beitragen.

Dies bedeutet, dass beispielsweise Großverbraucher in Zeiten geringen Energiebedarfs zeitweilig vom Netz genommen und die dadurch freiwerdenden Kapazitäten anderwertig eingesetzt werden. Dies hätte zur Folge, dass sowohl Regelungsverluste als auch der Regelungsaufwand verringert und somit Kosten gespart werden könnten. Es ist vorstellbar, dass Anlagen zur Kälteerzeugung aufgrund der thermischen Trägheit zu günstigen Zeiten die Temperatur zusätzlich senken können und zu ungünstigen Zeiten die Kühltätigkeit auf das Minimum herunterfahren. Hiermit ist also das Interesse begründet, das Lastverlagerungspotential in der Kälteerzeugung zu untersuchen.

2.2 Kälteerzeugung im Supermarkt: Potentiale und Besonderheiten

Im Mittel entfallen 60 % [4,7] des Stromverbrauchs in einem Supermarkt auf das Kühlen und Tiefkühlen von Produkten. Der durchschnittliche Verbrauch in einem Supermarkt liegt zwischen 300 MWh und 500 MWh im Jahr [7]. In den kommenden Jahren schätzt man das Wachstum des gesamten Kältebedarfs für Deutschland auf 100 % [13]. Der Anteil am Bedarf an elektrischer Energie vom Gesamtverbrauch eines Industrielandes für Kälteerzeugung in den Supermärkten wird in der Literatur für Australien mit 1 % [9] und in Schweden mit 2 % [2,4] angegeben. In Deutschland liegt der Verbrauch mit 6294 GWh für das Jahr 1999 bei rund $1,27\,\%$ [16]. Die Größenordnung des Verbrauchs für die Produktlagerung in Supermärkten und die Möglichkeit diesen zeitlich zu verschieben, macht die Supermärkte für die Regelung besonders interessant. Jedoch unterliegt der Einsatz bestimmten Beschränkungen und Randbedingungen.

Die Mindesthaltbarkeit für bestimmte Produkte kann nur durch Lagerung dieser Produkte in für sie festgelegtem Kühltemperaturbereichen garantiert werden. Dieser Temperaturbereich ist je nach Bedarf und Anwendung in Normalkühlung (NK) über 0°C und in Tiefkühlung (TK) unter 0°C unterteilt. Auf der nationalen und auch auf der internationalen Ebene existieren Auflagen, die die maximale Temperaturen bei der Lagerung von Nahrungsmitteln vorschreiben. Die fundamentale Verordnung ist die EG 853/2004. Aus der geht hervor, dass normalgekühlte Produkte im Temperaturbereich von 0 bis +8°C je nach Nahrungsmittel und tiefgekühlte Produkte mindestens bei −18°C gekühlt werden müssen.

Die Kälteerzeugung im Supermarkt kann räumlich generell in zwei Bereiche unterteilt werden, dem Verkaufsbereich und dem Warenlager. Außerdem kommen Kühleinheiten zum Einsatz, die einer Verbundkälteanlage angehören oder steckerfertig zur Verfügung stehen. Im Verkaufsbereich kommen überwiegend Kühltruhen im Tiefkühlbereich und Kühltruhen und -regale im Niederkühlbereich zum Einsatz. Zur Ausstattung zählen Geräte in offener sowie durch Türen verschließbarer Ausführung. Größtenteils werden die offenen Modelle außerhalb der Öffnungszeiten

durch Decken und Rollos zwecks Energieeinsparung verschlossen.

2.3 Objektorientierte Programmierung mit MATLAB®

Seit Ende des letzten Jahrhunderts herrscht in der Fachliteratur für Informatik die Meinung, dass der Einsatz von objektorientierten Techniken Programme hervorbringt, die im Vergleich *einfacher erweiterbar, besser testbar* und *besser wartbar* sind. Dabei wird ein Verfahren angewendet, nachdem große Systeme in kleinere Teile des Ganzen zerlegt werden. Programme lassen sich dadurch im Allgemeinen mit weniger Aufwand und kleinerer Fehlerwahrscheinlichkeit programmieren. Inspiriert durch die Vorgänge aus der realen Welt, werden die Abläufe durch operierende Objekte dargestellt, die Aufträge erledigen und vergeben können. Die wesentlichen Eigenschaften der objektorientierten Programmierung, kurz OOP, sind die Datenkapselung, die Polymorphie und die Vererbung.[1]

Allgemeine Erläuterungen

Klasse Eine Klasse ist ein Instrument der Programmierung zur Erfassung von charakteristischen Eigenschaften zusammenhängender Objekte. Die Definition der Strukturen der Objekte erfolgt durch Klassen.

Objekt Ein Objekt ist ein konkretes Exemplar einer Klasse.

Eigenschaften Eigenschaften sind Variablen, die für jedes Objekt existieren.

Methoden Methoden sind objektlokale Funktionen.

Datenkapselung Man spricht von Kapselung, wenn Objekte den Zugriff auf ihre Daten kontrollieren.

Polymorphie Können unterschiedliche Objekte auf eine gleiche Nachricht unterschiedlich reagieren, spricht man von Polymorphie.

Vererbung Die Vererbung ermöglicht durch Veränderung der bestehenden Klassen neue Klassen zu erstellen. Die grundlegenden Programmteile der bestehenden Klasse werden zwangsläufig übernomen.

Erläuterungen zur OOP-Syntax[2]

classdef Neue Klassen werden mit der Anweisung `classdef` eingeleitet. Der Name der Klasse wird unmittelbar nach der Anweisung eingetragen. Der Klassen-

[1] Ausführliche Informationen dazu findet man z.B. in [3], [11], [12] oder [18].

[2] Es werden nur Schlüsselwörter vorgestellt, die essentiell sind. Eine tiefgreifende Darstellung würde den Rahmen einer Studienarbeit bei Weitem übersteigen. Ausführliche Informationen dazu findet man z.B. in [15] oder `http://www.mathworks.com/help/`.

Block endet mit der Anweisung `end`.

properties Die Anweisung `properties` beginnt den Eigenschaften-Block. Abgeschlossen wird der Block mit `end`. Innerhalb einer Klasse können mehrere Blöcke existieren. Möchte man das Verhalten eines Eigenschaften-Blocks zusätzlich verändern, zum Beispiel neue Zugriffsarten oder Zugriffsrechte vergeben, ist der Einsatz von Attributen zweckmäßig.

```
properties (attribut1, attribut2, etc.)
end
```

methods Die Anweisung `methods` beginnt den Methoden-Block und mit `end` wird er geschlossen. Analog zu dem Eigenschaften-Block können in einer Klasse mehrere Methoden-Blöcke existieren. Möchte man das Verhalten eines Methoden-Blocks zusätzlich verändern, ist der Einsatz von Attributen zweckmäßig.

```
methods (attribut1, attribut2, etc.)
end
```

Klassen und Objekte mit MATLAB®

Klassen werden in MATLAB® mit Hilfe einer Datei mit der Endung *.m* definiert, die den selben Namen wie die Klasse hat. Im Quelltextbeispiel MATLAB® -Code 2.1 wird eine Beispielklasse **class_name** mit einem Eigenschaften-Block und einem Methoden-Block definiert. Die Datei muss demnach **class_name.m** heißen. In die erste Zeile der Datei kommt die Anweisung `classdef`, die die Klasse einleitet. Werden der Klasse zusätzliche Verhaltensmuster zugewiesen, folgt die Setzung der Attribute in Klammern. Anschließend kommt der Name der Klasse. Wird die neue Klasse eine durch Vererbung abgeleitete einer anderen Klasse, so folgt das Kleinerzeichen und der Name der Superklasse. In den anschließenden Zeilen werden der Eigenschaften-Block und der Methoden-Block definiert.

Objekte einer Klasse werden durch eine Konstruktor-Methode erzeugt. Diese Funktion wird im ersten Methoden-Block an erster Stelle definiert. Sie hat den selben Namen wie die Klasse. Über die Konstruktor-Funktion können bereits bei der Erzeugung der Objekte Werte an ausgewählte Attribute übergeben werden. Erfolgt der Aufruf einer Konstruktor-Funktion einer Klasse fehlerfrei, so ist das Ergebnis des Aufrufes ein Objekt dieser Klasse. Der Aufruf eines Konstruktors unterscheidet sich nicht von dem Aufruf gewöhnlicher MATLAB® -Funktionen. Im folgenden Quelltextbeispiel wird im Command-Window durch den Aufruf der Konstruktor-Funktion ein Objekt **object** der im MATLAB® -Code 2.1 vorgestellten Klasse erzeugt.

```
>> object = class_name(input_data);
```

MATLAB® -Code 2.1: Beispiel Klassendefinition

```
classdef (Attributes) class_name < super_class % class definition
    properties (Attributes) % first property block
        PropertyName = [];
    end % end of properties block
    % additionally here can be another properties block with
    % specifying by another Attributes
    methods (Attributes)
        function obj = class_name(obj,a) % consturctor
            obj.PropertyName = a;
        end
        function [X] = second_function(obj)
            X = obj.PropertyName + 1;
        end
    end
    % additionally here can be another methods block with
    % specifying by another Attributes
end % end of classdef block
```

Methoden, die applikationsspezifische Operationen mit einem Datensatz durchführen sollen, werden nach der Konstruktor-Funktion definiert. Die meisten dieser Funktionen nutzen das Objekt als Eingabeargument, zum Beispiel: `second_function(obj)`. Der Zugriff auf die Variablen der Objekteigenschaften erfolgt durch Referenzierung auf das Objekt. Im Quelltextbeispiel MATLAB® -Code 2.1 wird das wie folgt vorgestellt: `obj.PropertyName`. Wenn ein Zugriff auf Objektfunktionen erlaubt ist, kann darauf wie folgt zugegriffen werden:

```
>> function_return = object.second_function;
```

Die Referenz auf den Namen des Objektes muss stets erfolgen.

2.4 Einführung in die Kältetechnik

An dieser Stelle wird ein knapper Überblick über die mathematischen und physikalischen Zusammenhänge gegeben, die bei der Entwicklung eines Modells einer Kälteanlage in einem Modellsupermarkt zwingend beachtet werden müssen.

In den Supermärkten werden Kühlgeräte zur Lagerung der Ware bis zum Verkauf an den Endkunden eingesetzt. Um die Haltbarkeit dieser Ware für den Mindestzeitraum zu gewährleisten, wird sie bei niedrigen Temperaturen gehalten. Körper mit unterschiedlicher Temperatur sind bestrebt, wenn sie thermisch von einander nicht vollkommen isoliert sind, durch gegenseitige Wechselwirkung ihre Temperaturen anzugleichen. Infolge dessen wird ein Wärmegleichgewicht erreicht. Der natürliche Wärmefluss findet selbstständig immer vom Körper mit der höheren

Temperatur in die Richtung des Körpers mit der niedrigeren Temperatur statt. Um eine negative Temperaturänderung herzustellen und diese auch zu halten, muss die eindringende Wärmeenergie ständig in derselben Höhe abgeführt werden, damit die Temperatur konstant bleibt. Diese Energiemenge pro Zeiteinheit wird als Kälteleistung bezeichnet. Eine Abweichung von dieser Menge führt zum Steigen der Temperatur, wenn weniger und zum Sinken der Temperatur wenn mehr abgeführt wird. Um diesen Kühlkreislauf aufrecht zu erhalten, muss Leistung aufgewendet werden[3].

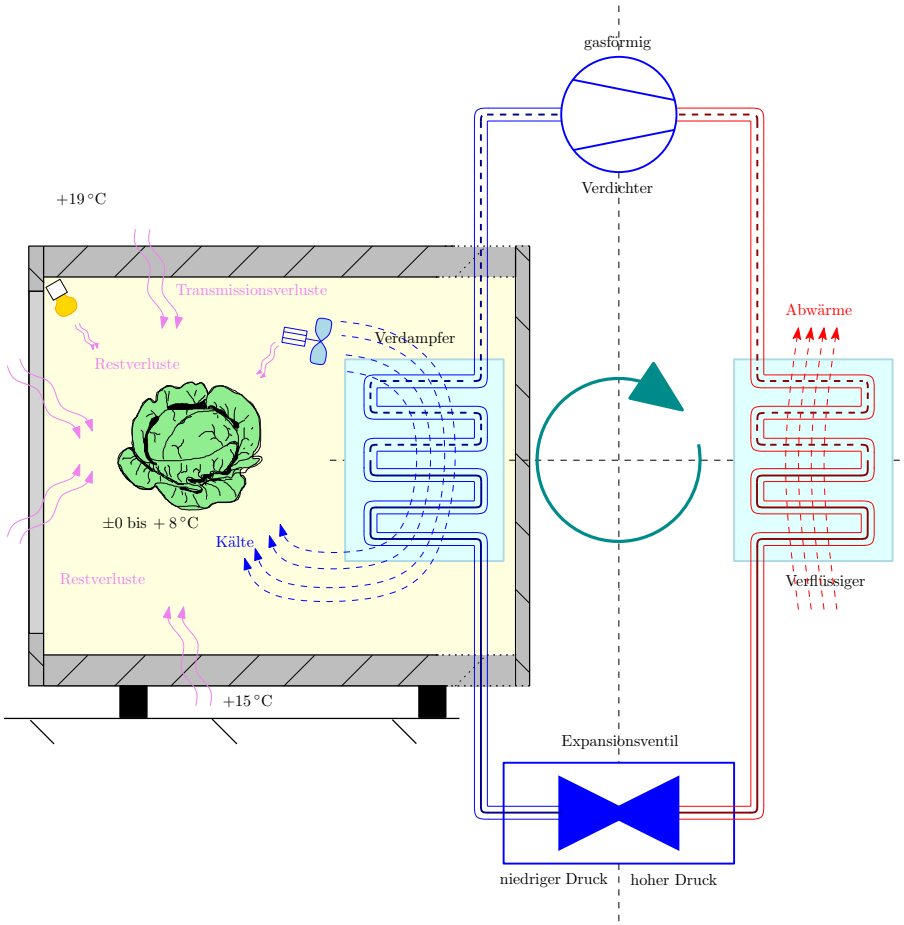

Abbildung 2.1: Prinzipdarstellung Kompressionskälteanlage

In der Abbildung 2.1 auf der Seite 7 ist die Prinzipdarstellung der weitverbreiteten Kompressionskältemaschine zu sehen. Die Ware, die symbolisch durch einen Kohlkopf dargestellt ist, wird mit einer Temperatur zwischen ±0 und +8°C gelagert. Der Temperaturunterschied zwischen dem Innen- und Außenbereich ist maßgebend für den Verlust an Kälte (Wärmeausgleich). Der Ausgleich führt dazu, dass Kältenergie für den Lagerungs- und Kühlungsprozess verloren geht.

[3] Eine detailierte Beschreibung dieser Prozesse in einer Kompressionskälteanlage und Spezifikation ist nicht Gegenstand dieser Arbeit. Ausführliche Informationen dazu findet man z.B. in [2,10,14].

Die Aufnahme der Wärmeenergie und der spezifischen Wärmekapazität dieser Substanzmasse ergibt die Temperaturdifferenz Δt.

$$\Delta t = \frac{Q}{m \cdot c} \tag{2.1}$$

Δt Temperaturdifferenz in Kelvin K

Q Verlust an Kälte in in kJ

m Substanzmasse zur Aufnahme der Kälteenergie in kg

c Spezifische Kältekapazität (Wärmekapazität) der Substanzmasse in $\frac{\text{kJ}}{\text{kg}\cdot\text{K}}$

Die installierte Kälteleistung multipliziert mit der täglichen Betriebszeit muss gleich dem stündlichen Kältebedarf multipliziert mit der Tagesstundenzahl sein.

$$\dot{Q}_0^{)} = \frac{24}{\tau_B} \cdot \dot{Q}_0 \tag{2.2}$$

$\dot{Q}_0^{)}$ Installierte Kälteleistung in kW

τ_B Tägliche Betriebszeit in h

$\dot{Q}_0$ stündlicher Kälteleistungsbedarf in kW

stündlicher!!! überall

Die Transmissionswärmeleistung wird aus der Multiplikation der Fläche der wärmeübertragenden Wände mit ihrem spezifischen Wärmedurchgangskoeffizient und der Temperaturdifferenz zwischen der Kühlraumtemperatur und der Umgebungstemperatur berechnet.

$$\dot{Q}_{Tr} = A \cdot k \cdot \Delta t \tag{2.3}$$

$\dot{Q}_{Tr}$ Transmissionswärmeleistung in kW

A Fläche in m^2

k Wärmedurchgangskoeffizientin $\frac{\text{W}}{\text{m}^2\cdot\text{K}}$

e zuviel

Δt Temperaturdifferenz in K

Der Zusammenhang zwischen der aufgewendeten elektrischen Antriebsleistung P eines Verdichters in einer Kompressionskälteanlage und der genutzten stündlichen Kälteleistung $\dot{Q}_0$ wird durch die Kältezahl ε wiedergegeben.

$$\varepsilon = \frac{\dot{Q}_0}{P} \tag{2.4}$$

ε Leistungszahl (einheitenlos)

$\dot{Q}_0$ stündliche Kälteleistungsbedarf in kW

P elektrische Verdichterantriebsleistung in einer Kompressionskälteanlage in kW

Die Öffnungszeit des Supermarkts hat einen spürbaren Einfluss auf die Größe der Kälteverluste. In der Literatur wird der Nachtverbrauch mit 10 % bis 20 % des Tagesverbrauchs angegeben [5]. In der Nacht fallen keine zusätzlichen Verluste an, zum Beispiel durch Licht, Körperwärme oder Türöffnungszeiten, sodass nur Transmissionsverluste bei der Berechnung beachtet werden. Leistungsbedarf definiert die Menge an Leistung, der benötigt wird, um den Verlust an Kälte zu ersetzen. Die Abweichung zwischen Tagesbedarf und Nachtbedarf an Leistung wird im weiteren Verlauf Tagesmehrbedarf genannt.

$$\dot{Q}_{Nacht} = \dot{Q}_{Tr} \tag{2.5}$$

$\dot{Q}_{Nacht}$ Leistungsbedarf in der Nacht in kW

$\dot{Q}_{Tr}$ Transmissionswärmeleistung in kW

Der Kälteleistungsbedarf am Tag ergibt sich aus der Summe des Tagesmehrbedarfs und der Transmissionswärmeleistung. Aus Gründen der Vereinfachung wird der Tagesmehrbedarf als weitgehend konstant angenommen.

$$\dot{Q}_{Tag} = \dot{Q}_{mehr} + \dot{Q}_{Tr} \tag{2.6}$$

$\dot{Q}_{Tag}$ Leistungsbedarf am Tag in kW

$\dot{Q}_{mehr}$ Tagesmehrbedarf am Tag in kW

$\dot{Q}_{Tr}$ Transmissionswärmeleistung in kW

Die Verlustkältememenge im Kühlbereich ist aufgrund der obengenannten Gründe je nach Tageszeit unterschiedlich.

$$Q_v = \begin{cases} 1h \cdot \dot{Q}_{Nacht}, & \text{außerhalb der Öffnungszeiten} \\ 1h \cdot \dot{Q}_{Tag}, & \text{innerhalb der Öffnungszeiten} \end{cases} \tag{2.7}$$

Q_v Verlustkältemenge in kJ

$\dot{Q}_{Tag}$ Leistungsbedarf am Tag in kW

$\dot{Q}_{Nacht}$ Leistungsbedarf in der Nacht in kW

h Zeiteinheit für Stunden

Ist für eine Kälteanlage der stündliche Kälteleistungsbedarf bekannt, so wird der Tagesmehrbedarf an Kälteleistung ermittelt, indem vom Produkt des Kälteleistungsbedarfes mit dem Faktor für Kältebedarfsabsenkung die mittlere Transmissionswärmeleistung abgezogen wird.

$$\dot{Q}_{mehr} = \dot{Q}_0 \cdot K - \overline{\dot{Q}}_{Tr} \tag{2.8}$$

$\dot{Q}_{mehr}$ Tagesmehrbedarf in kW

$\dot{Q}_0$ stündliche Kälteleistungsbedarf in kW

K Faktor für Kältebedarfsabsenkung (einheitenlos)

$\overline{\dot{Q}}_{Tr}$ mittlere Transmissionswärmeleistung in kW

Die Berechnung der mittleren Transmissionswärmeleistung erfolgt durch die Multiplikation der Differenz zwischen der Umgebungstemperatur und der mittleren Kühlraumtemperatur mit dem Wärmedurchgangskoeffizienten und der Fläche der wärmeübertragenden Wände.

$$\overline{\dot{Q}}_{Tr} = A \cdot k \cdot \left(t_{amb} - \overline{t}_{KR}\right) \tag{2.9}$$

$\overline{\dot{Q}}_{Tr}$ mittlere Transmissionswärmeleistung in kW

A Fläche in m^2

k Wärmedurchgangskoeffiziente in $\frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}}$

t_{amb} Umgebungstemperatur in ° C

$\overline{t}_{KR}$ mittlere Kühlraumtemperatur in ° C

Ist der Kälteleistungsbedarf nicht bekannt, wie zum Beispiel bei steckerfertigen Geräten, kann der Mehrbedarf am Tag über den Wert Verdichterarbeit pro 24 Stunden $W_{Verd_{24}}$ für den gesamten Kälteverbraucher ermittelt werden.

Das Produkt aus dem spezifischen Energieverbrauch mit dem Faktor für Kältebedarfsabsenkung, dem Verdichteranteil ergibt die Verdichterarbeit.

$$v_{mod} = K \cdot v \tag{2.10}$$

$$W_{Verd_{24}} = W_{spez_{24}} \cdot v_{mod} \tag{2.11}$$

$W_{Verd_{24}}$ Verdichterarbeit pro 24 Stunden in $\frac{\mathrm{kWh}}{24\mathrm{h}}$

$W_{spez_{24}}$ spezifischer Energieverbrauch pro 24 Stunden in $\frac{\text{kWh}}{\text{24h}}$

K Faktor für Kältebedarfsabsenkung (einheitenlos)

v Verdichteranteil (einheitenlos)

v_{mod} modifizierter Verdichteranteil (einheitenlos)

Im Folgenden muss die Verdichterarbeit zum Ersetzen der zusätzlichen Kälteenergieverlustmenge in der Öffnungszeit berechnet werden, die in einem weiteren Schritt in Tagesmehrbedarf $\dot{Q}_{mehr}$ umgewandelt wird.

$$W_{mehr} = W_{Verd_{24}} - \frac{\overline{\dot{Q}}_{Tr}}{\varepsilon} \cdot 24 \tag{2.12}$$

W_{mehr} Verdichterarbeit zum Ersetzen der zusätzlichen Kälteenergieverluste Q_{mehr} in der Öffnungszeit in kWh

$W_{Verd_{24}}$ Verdichterarbeit pro 24 Stunden in $\frac{\text{kWh}}{\text{24h}}$

$\overline{\dot{Q}}_{Tr}$ mittlere Transmissionswärmeleistung in kW

ε Leistungszahl (einheitenlos)

Die Umrechnung von W_{mehr} in $\dot{Q}_{mehr}$ erfolgt mit Hilfe der Leistungszahl ε. Die Verdichterarbeit W_{mehr} wird mit ε multipliziert. Es wird angenommen, dass der Mehrbedarf in den rund zwölf Stunden der Öffnungszeit entsteht.

$$\dot{Q}_{mehr} = \frac{W_{mehr}}{12} \cdot \varepsilon \tag{2.13}$$

$\dot{Q}_{mehr}$ Tagesmehrbedarf in kW

W_{mehr} Verdichterarbeit zum Ersetzen der zusätzlichen Kälteenergieverluste Q_{mehr} in der Öffnungszeit in kWh

ε Leistungszahl (einheitenlos)

Kapitel 3

Aufgabenstellung

Die Intention der Arbeit, ist der objektorientierte Entwurf eines MATLAB®-Programms zur Simulation des variablen Lastverhaltens von Kältelasten mit Kältespeichern im Energieversorgungsnetz.

Es sollen ausschließlich Kälteanlagen modelliert werden, die in den Supermärkten zum Einsatz kommen. Das Ergebnis der Simulation soll der Verbrauch einer variabel geführten Supermarktkette auf der Basis der einzelnen in den Supermärkten eingesetzten Kälteanlagen sein.

Verbrauchswerte in einem Lastflussoptimierungsprogramm zu verwenden.

Explizite Anforderungen

Folgende explizite Anforderungen sind an das Programm gestellt:

- Die Speicherung der für das Beschreiben der Modelle erforderlichen Parameter findet in einer Konfigurationsdatei statt.
- Der elektrische Energieverbrauch wird auf Grund der Modellparameter der Kältelasten sowie des gefahrenen Einsatzes der Kältelasten berechnet.
- Die Topologie des elektrischen Netzwerkes werden berücksichtigt.
- Die Möglichkeit einer eindeutigen Zuweisung der Verbrauchswerte für folgende Verursacher wird realisiert:
  - Daten für jede Kälteanlage werden einzeln gespeichert, wenn der Verbrauch mehrerer Anlagen simuliert wird.
  - Daten für jede Supermarktkette werden einzeln gespeichert, wenn der Verbrach mehrerer Supermarktketten simuliert wird.
  - Erfolgt die Simulation des Energieverbrauchs für ein Mehrknotennetz, so werden die Daten für jeden Knoten, einzeln gespeichert.

Kapitel 4

Modellierung

Inhaltsangabe

Der durchschnittliche Energieverbrauch der Kälteanlagen je Supermaktkette kann aufgrund der technischen Ausführung unterschiedlich sein. Der Energieverbrauch entsteht an definierten Punkten im Netz. An den einzelnen Knotenpunkten können mehrere Kältelasten angeschlossen sein.

4.1 Erstellung des OOP-Modells

In der Abbildung 4.1 wird ein einfaches Energieversorgungsnetz mit vier Knoten dargestellt. Am Knoten eins ist eine regenerative elektrische Energiequelle, in diesem Fall ein Windpark, angeschlossen. Weitere konventionelle elektrische Energiequellen befinden sich an den Knoten zwei und drei. Die passiven Lasten befinden sich am Knoten zwei und vier. Der Kältespeicher ist am Knoten zwei angeschlossen. Im Bild wird der Kältespeicher Supermarkt durch ein Gebäude mit Einkaufswagen dargestellt. Die Knoten sind untereinander durch Leitungen verbunden.
In der Realität kann ein Energieversorgungsnetz durch die Variation der Knotenzahl und die Vermaschung eine beliebig komplizierte Form aufweisen.

In Anbetracht der im Abschnitt 2.3 vorgestellten Verfahren erscheint der Ansatz der OOP bei der Erstellung eines Programms zur Simulation einer Kälteslast mit Kältespeicher im Energieversorgungsnetz auf der Basis des vorangegangenen Beispiels (Abbildung 4.1 auf der Seite 14) als sinnvoll. In der Abbildung 4.2 auf der Seite 15 wird in Form eines Klassendiagramms das fertige Klassen-Modell dargestellt und die Abhängigkeit unter Klassen visualisiert.

Das Ergebnis der Abstraktion sind vier Klassen. In der Klasse **Refrigerator** sind Eigenschaften und Methoden zusammengefasst, die das Modell einer Kältelast beschreiben. Die explizite Erklärung der.

hier ein Überleitungssatz

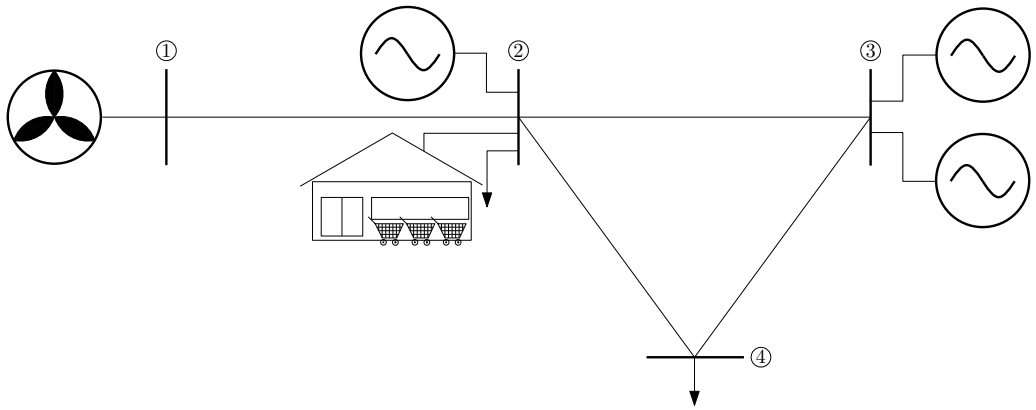

Abbildung 4.1: Vier Knoten Beispiel

Refrigerator

An dieser Stelle werden die Eigenschaften und die Methoden der Klasse **Refrigerator**explizit erklärt.

trennung!

Attribute

fridge_detector Art der Anlage. Die Berechnung der Verluste der Verbundanlagen unterscheidet sich von der der steckerfertigen Geräte.

energy_consumption_day Höhe des durchschnittlichen elektrischen Energieverbrauchs.

epsilon Leistungszahl.

compressor_quotient Anteil des Verdichters am Gesamtverbrauch der Kühlanlage.

trennung

cooling_power Kälteleistungsbedarf (Herstellerangaben).

temperatur_outside_fridge $1 \times M$-Array mit Außentemperaturwertet für die Umgebung der Kälteanlage. M stimmt mit der Anzahl der Wände überein, für die ihre Fläche bekannt ist.

mass_stored $1 \times N$-Array mit den Massewerten, der zur Kühlung vorhandenen Substanzmasse. N stimmt mit der Anzahl der Massen überein, für die die spezifische Wärmekapazität unterschiedlich ist.

specific_heat_capacity $1 \times N$-Array mit den Werten für spezifische Wärmekapazitäten der Substanzmassen.

cooling_room_temperature_min Niedrigste Temperatur, die durch Kühlung erreicht werden darf.

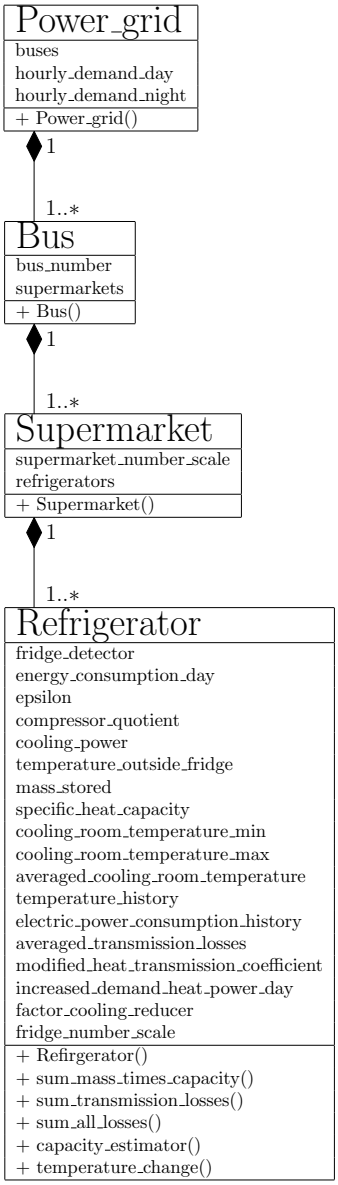

Abbildung 4.2: Klassendiagramm Modellkonstrukt

cooling_room_temperature_max Höchste Temperatur, die durch Kühlung erreicht werden darf.

averaged_cooling_room_temperature Durchschnittliche Temperatur im Normalbetrieb.

temperature_history $n \times m$-Array mit $(n, m) \in \mathbb{Z}_0^+$, wobei n für die Anzahl der Werte des berechneten Temperaturverlaufs und m für die Anzahl der simulierten Tage steht. Die Werte sind das Ergebnis der Berechnung der Funktion **main_equation**. Eine Spalte des Arrays stellt den Temperaturverlauf an genau einem Tag dar.

electric_power_consumption_history $n \times m$-Array mit $(n, m) \in \mathbb{Z}_0^+$, wobei n für die Anzahl der Werte des berechneten Verbrauchs und m für die Anzahl der simulierten Tage steht. Die Werte sind das Ergebnis der Berechnung der Funktion **main_equation**. Eine Spalte des Arrays stellt den Verlauf des berechneten Verbrauches an genau einem Tag dar.

averaged_transmission_losses Summe aller durchschnittlichen Transmissionsverluste im Normalbetrieb (vgl. Gleichung (2.9) auf der Seite 10).

modified_heat_transmission_coefficient $1 \times M$-Array mit Werten aus der elementeweisenMultiplikation des Vektors mit Werten der spezifischen Wärmedurchgangskoeffizienten mit dem Vektor mit Werten für die Durchgagnsflächen (vgl. Gleichung (2.10) auf der Seite 10).

trennung

increased_demand_heat_power_day Mehrbedarf am Tag, aufgrund der zusätzlichen Verluste durch Licht, Menschenkörperwärme, Türöffnungszeiten und weiteren Einflüsse. Die Berechnung erfolgt nach einer Auswahl zwischen der Gleichung (2.8) und der Gleichung (2.11) bis Gleichung (2.13) wie auf der Seiten 10 und 11 beschrieben.

factor_cooling_reducer Faktor für Kältebedarfsabsenkung infolge der Luftfeuchtigkeit und der Umgebungstemperatur.

fridge_number_scale Anzahl der Anlagen mit der identischen technischen Ausführung, der Beladung und dem Fahrplan.

Methoden

Refrigerator(fridge_config,number_steps,days) Konstruktor der Klasse. Bei dem Aufruf dieser Funktion werden folgende Argumente übergeben:

fridge_config 1×2-Cell Array mit Modellparametern[1].

number_steps Anzahl der Zeitschritte die einen Tag beschreiben.

days Anzahl der zu simulierenden Tage.

Bei der Initialisierung eines Objektes werden einem Teil der oben vorgestellten Attribute der Klasse **Refrigerator** Werte aus dem **fridge_config** -Array direkt übergeben. Andere Werte werden einmalig von der Konstruktor-Methode berechnet oder im Laufe der Simulation verändert.

Folgende Attribute werden einmalig bei der Initialisierung berechnet:

averaged_transmission_losses Summe aller durchschnittlichen Transmissionsverluste im Normalbetrieb (vgl. Gleichung (2.9) auf der Seite 10).

[1] Die Form des Arrays ist äquivalent zu genaue einer Zeile der in MATLAB®-Code 4.2 auf der Seite 30 vorgestellten Arrays, die die Modelle der Supermärkte abbilden.

modified_heat_transmission_coefficient $1 \times M$-Array mit Werten aus der elementeweisen Multiplikation des Vektors mit Werten der spezifischen Wärmedurchgangskoeffizienten mit dem Vektor mit Werten für die Durchgagnsflächen (vgl. Gleichung (2.10) auf der Seite 10).

increased_demand_heat_power_day Mehrbedarf am Tag , aufgrund der zusätzlichen Verluste durch Licht, Menschenkörperwärme, Türöffnungszeiten und weiteren Einflüsse. Die Berechnung erfolgt nach einer Auswahl zwischen der Gleichung (2.8) und der Gleichung (2.11) bis Gleichung (2.13) wie auf der Seiten 10 und 11 beschrieben.

Folgende Attribute werden im Laufe der Simulation verändert:

temperature_history $n \times m$-Array mit $(n, m) \in \mathbb{Z}_0^+$, wobei n für die Anzahl der Werte des berechneten Temperaturverlaufs und m für die Anzahl der simulierten Tage steht. Die Werte sind das Ergebnis der Berechnung der Funktion **main_equation**. Eine Spalte des Arrays stellt den Temperaturverlauf an genau einem Tag dar.

electric_power_consumption_history $n \times m$-Array mit $(n, m) \in \mathbb{Z}_0^+$, wobei n für die Anzahl der Werten des berechneten Verbrauchs und m für die Anzahl der simulierten Tage steht. Die Werte sind das Ergebnis der Berechnung der Funktion **main_equation**. Die Werte sind das Ergebnis der Berechnung der Funktion **main_equation**. Eine Spalte des Arrays stellt den Verlauf des berechneten Verbrauches an genau einem Tag dar.

Folgende Attribute sind auf Grund der Annahmen dieser Studie konstant, können aber bei der bei einer Modifizierung der Annahmen im Laufe der Simulation verändert werden:

temperature_outside_fridge Befindet sich der Kältespeicher außerhalb geschlossener Räume und im direkten Kontakt mit der Außentemperatur, muss die Temperaturänderung die im Tagesverlauf in Abhängigkeit von der Jahreszeit entsteht, beachtet werden.

mass_stored Wenn nicht mehr Angenommen wird, dass die Substansmasse im Supermarkt im Laufe des Tages konstant ist, sondern mit dem Konsumverhalten der Bevölkerung in Abhängigkeit steht, muss eine Anpassung erfolgen.

sum_mass_times_capacity() Der Rückgabewert dieser Funktion ist die berechnete Summe aus elementeweisen Multiplikation aller Substansmassen mit der jeweiligen spezifischen Wärmekapazitäten. Der Funktion werden keine explizite Argumente übergeben. Die Funktion greift direkt auf die Attribute **specific_heat_capacity** und **mass_stored** des Objektes zu.

sum_transmission_losses() Es gibt zwei Rückgabewerte dieser Funktion. Der erste Wert ist die Summe der Transmissionsverluste (vgl. Gleichung (2.9) auf

der Seite 8) durch alle Durchgangsflächen für den aktuellen Zeitpunkt und die aktuelle Kühlraumtemperatur. Desweiteren wird die Summe der Transimissionsverluste berechnet, die bei der oberen Grenze des zugelassenen Temperaturbereiches entstehen. Die Kenntnis dieser Verluste ist wichtig um die Zeitspanne abschätzen zu können, bis obere Temperaturgrenze erreicht wird (vgl. Gleichung (4.2) auf der Seite 22).

Die Übergabe folgender Argumente an die Funktion ist erforderlich:

number_steps Laufvariable. Gesamtzahl der zu simulierenden Zeitschrite für ein Tag.

count_number_steps Der aktuelle Wert von **number_steps**.

count_number_day Der aktuelle Wert von der Laufvariable **number_days**.

sum_all_losses() Es gibt zwei Rückgabewerte dieser Funktion. Der erste Rückgabewert beinhaltet die Summe aller Verluste an Kälte (vgl. Gleichung (2.7) auf der Seite 9). Außerhalb der Öffnungszeiten entfällt der Anteil an zusätzlichen Verlusten, die durch Licht, offene Türen und etc. anfallen. Durch eine alternative Verzweigung, deren Bedingung die Zugehörigkeit des aktuellen Wertes der Laufvariable **count_number_steps** zu den Öffnungszeiten ist, wird die Berücksichtigung der zusätzlichen Verluste sichergestellt. Der zweite Ruckgabewert ist die Zeitspanne bis zum Erreichen der kritischen maximalen Temperatur (vgl. Gleichung (4.2) auf der Seite 22).

Die Übergabe folgender Argumente an die Funktion ist erforderlich:

number_steps Laufvariable. Gesamtzahl der zu simulierenden Zeitschrite für ein Tag.

count_number_steps Der aktuelle Wert von **number_steps**.

count_number_day Der aktuelle Wert von der Laufvariable **number_days**.

number_days Laufvariable. Gesamtzahl der zu simulierenden Tage.

capacity_estimator() Die Funktion hat einen Rückgabewert. Berechnet wird die maximal abzuführende Wermeenergiemenge Q_{max} (vgl. die Gleichung (4.7) auf der Seite 26).

Die Übergabe folgender Argumente an die Funktion ist erforderlich:

count_number_steps Der aktuelle Wert von **number_steps**.

count_number_day Der aktuelle Wert von der Laufvariable **number_days**.

Q_losses Wert der aktuellen Verlustkältemenge als Ergebnis der Funktion **sum_all_losses()** (vgl. Gleichung (2.7) auf der Seite 9).

temperature_change() Die Funktion hat einen Rückgabewert. Berechnet wird die Temperatur der Substanzmasse, die durch Kühlung am Ende eines Zeitschrittes erreicht wird (vgl. die Gleichung (4.6) auf der Seite 24).

Die Übergabe folgender Argumente an die Funktion ist erforderlich:

number_steps Laufvariable. Gesamtzahl der zu simulierenden Zeitschrite für ein Tag.

count_number_steps Der aktuelle Wert von **number_steps**.

count_number_day Der aktuelle Wert von der Laufvariable **number_days**.

Q_losses Wert der aktuellen Verlustkältemenge als Ergebnis der Funktion **sum_all_losses()** (vgl. Gleichung (2.7) auf der Seite 9).

Alle Klassen haben einen unterschiedlichen Status. Die Klasse **Power_gird** vertritt die Rolle des ganzen Systems, also des Energieversorgungsnetzes, welches aus charakteristischen Teilen, den Knoten (**Bus**) besteht. Die Existenz eines Knotens außerhalb eines Netzes ergibt keinen Sinn. Dieser Zusammenhang zwischen Klassen wird im Klassendiagramm Abbildung 4.2 durch eine Linienverbindung/Komposition verdeutlicht. Auf der Seite des Ganzen endet die Linie mit einem ausgefüllten Rhombus. Das komplette Modell ist nach diesem Prinzip aufgestellt. Das kleinste Element des Ganzen stellt die Kälteanlage (**Refrigerator**) dar.

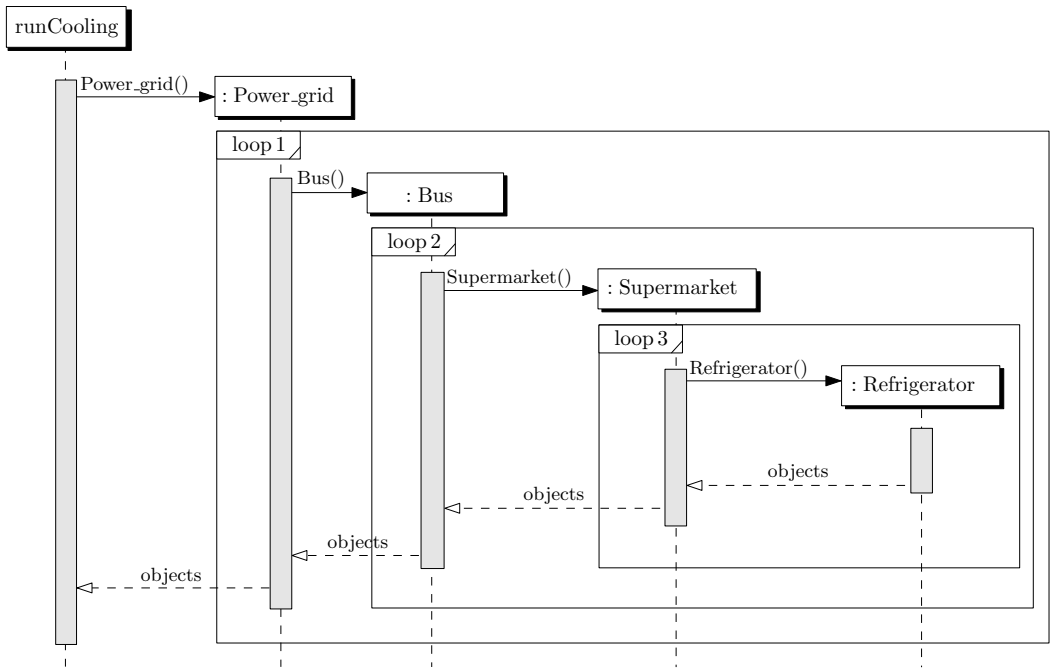

Abbildung 4.3: Sequenzdiagramm Modellkonstrukt

Auf Grund des Modells wird das elektrische Verhalten einer Supermarket-Kältelast im Energieversorgungsnetz durch miteinander kooperierende Objekte

dargestellt. Wird eine Funktion aufgerufen, die den Start der Simulation veranlasst, müssen die Objekte aus der Konfigurationsdatei erzeugt werden.

In der Abbildung 4.3 auf der Seite 19 wird mit Hilfe eines Sequenzdiagramms der Nachrichtenaustausch hervorgehoben, der bei der Erstellung der Objekte durchlaufen wird. Auf der Zeitachse, die senkrecht von oben nach unten verläuft, wird mit Hilfe der Pfeile der Zugriff auf die Methoden der Objekte dargestellt. Ist eine Methode des Objektes für einen Zeitraum aktiv, wird dieser Zustand durch einen grauen Balken auf dem Zeitstrahl des zugehörigen Objektes visualisiert. Der Aufruf der Funktionen der Objekte wird durch einen Pfeil mit ausgefüllter Spitze dargestellt. Der Pfeil geht vom Zeitstrahl eines Objekts aus, der die Funktion aufruft und endet mit der Spitze im Objekt, zu dem diese Funktion gehört. Der Name der aufgerufenen Funktion steht über dem Pfeil. Wird von dieser Funktion etwas zurückgegeben, so ist der Pfeil gestrichelt. Die Rückgabewerte stehen über dem Pfeil.

In der Funktion **runCooling**, die den Start der Simulation veranlasst und steuert, wird der Konstruktor der Klasse **Power_grid** aufgerufen. Als Ergebnis des Aufrufes bekommt die Funktion **runCooling** ein Objekt der Klasse **Power_grid** mit allen Unterobjekten, die den zu simulierenden Fall beschreiben. Die Konstruktor-Funktionen der Klassen, die einen untergeordneten Status besitzen, werden bei der Erstellung der übergeordneten Objekte aufgerufen. Die Anzahl der erzeugten Objekte hängt davon ab, wie oft die Konstruktor-Funktion der jeweiligen Klasse aufgerufen wird. Diese Zahlen werden in Konfigurationsdateien abhängig von dem untersuchten Fall festgelegt. Die ausführliche Erklärung der Konfigurationsdateien findet im Abschnitt 4.2 statt.

Bereitstellung der Leistung zur Kälteerzeugung

Der augenblickliche Gleichgewicht zwischen dem Kältebedarf und der zur Verfügung stehenden Kälteleistung sichert die Einhaltung der Soll-Temperatur der Kühlgüter. Werden die Kälteanlagen als Energiespeicher zur Regelung eingesetzt, so führt das, auf Grund der fluktuierenden zur Verfügung stehenden Kälteleistung, zu Temperaturschwankungen. Die engen Grenzen des zulässigen Temperaturbandes dürfen nicht überschritten werden.

Verschiedene Konzepte für den Einsatz der Kälteanlagen als Energiespeicher im Energieversorgungsnetz sind denkbar. Im folgenden Abschnitt wird die Implementierung eines möglichen Kooperationskonzeptes zwischen einer Supermarktkette und einem Windpark vorgestellt.

Die prognostizierten und die tatsächlich vom Windpark produzierten elektrischen Windleistungsdaten der Vattenfall-Regelzone (heute 50Hertz-Regelzone) aus dem Jahr 2005 werden als Grundlage-Datensatz verwendet. Im Datensatz sind Stundenwerte von Windleistung gespeichert, weshalb in diesem Fall auch von Energie gesprochen werden kann.

Der Windpark ist bestrebt, die Verpflichtung über die Bereitstellung von Leistung beim Netzbetreiber zu erfühlen. Abweichungen, die aufgrund der fehlerhaften Windprognosen entstehen, sollen durch die Kooperation mit Supermarktketten aus-

geglichen werden.

Es wird angenommen, dass der Prognosefehler von der Windleistungsvorhersage für 24 Stunden den 15% entspricht. Der Windparkbetreiber meldet beim Netzbetreiber die untere Grenze an, also 85% von der Windleistungsvorhersage. Der Supermarkt besitzt einen stündlichen Bedarf, der unbedingt gedeckt werden muss. Ist der stündliche Bedarf größer als 15% der Prognose, wird die Differenz aus dem Netz gezogen. Diese Abweichung wird beim Netzbetreiber zur Erstellung des Lastfahrplanes mitgeteilt. Gibt es eine Abweichung von der Prognose, so versucht der Supermarkt diese zu absorbieren.

Die vom Windparkbetreiber beim Netzbetreiber anzumeldende Leistung wird berechnet, indem von der Windleistungsvorhersage die 85% bestimmt werden oder der stündliche Bedarf des Supermarktes abgezogen wird, wenn die 15% kleiner als der stündliche Bedarf sind.

$$b = 0.85 \cdot P_{DA}$$

oder

$$b = P_{DA} - P_{dem}$$

b Leistung, die vom Windparkbetreiber beim Netzbetreiber angemeldet wird in MW

P_{DA} Vorhersage der Windleistung (day ahead) in MW

P_{dem} stündlicher Leistungsbedarf des Supermarkts (demand) MW

Die vom Windparkbetreiber dem Supermarkt versprochene Leistung wird berechnet, indem von der Windleistungsvorhersage die 15% bestimmt werden oder nur der stündliche Bedarf, wenn die 15% kleiner als der stündliche Bedarf sind.

$$a = 0.15 \cdot P_{DA}$$

oder

$$a = P_{dem}$$

a Leistung, die vom Windparkbetreiber dem Supermarkt versprochen wird in MW

P_{DA} Vorhersage der Windleistung (day ahead) in MW

P_{dem} stündlicher Leistungsbedarf des Supermarkts (demand) MW

Die Leistung, die insgesamt den Kälteanlagen zur Verfügung steht, berechnet sich nach der Gleichung (4.1). Der Ausdruck $P_{RT} - b$ bedeutet die überschüssige Windleistung. Der Ausdruck $P_{dem} - a$ bedeutet die vom Supermarkts beim Netzbetreiber nicht abgemeldete Leistung.

$$P_{load} = P_{RT} - b + P_{dem} - a \tag{4.1}$$

P_{load} vorhandene Leistung zum Laden in MW

P_{RT} Windleistungsdaten. Windleistung, die tatsächlich vom Windpark generiert wurde in MW

b Leistung, die vom Windparkbetreiber beim Netzbetreiber angemeldet wird in MW

P_{dem} stündlicher Leistungsbedarf des Supermarkts (demand)

a Leistung, die vom Windparkbetreiber dem Supermarkt versprochen wird in MW

Implementierung

Die Implementierung des Kooperationskonzeptes erfolgt in einer neuen Klasse **Cooling_strategy**. Diese Klasse hat die Aufgabe, jeder Kühlstelle in jedem Supermarkt im Netz die aufgrund des Kooperationskonzeptes und der Temperaturbeschränkungen optimale Menge an Kälteleistung zuzuweisen. Die zur Verfügung stehende Leistung wird auf alle Kühlstellen gleichmäßig aufgeteilt, bis keine Leistung mehr vorhanden ist oder die Minimaltemperatur erreicht wird. Alle Kühlstellen sollen zu jeder Stunde mindestens soviel Leistung zugewiesen bekommen, dass genau diese eine Stunde überbrückt werden kann, ohne dass die Höchsttemperatur überschritten wird. Die Zeit bis zur Höchsttemperatur darf also nie unter eine Stunde fallen. Um diese Zeitbeschränkung zu erfüllen, muss ermittelt werden, ob die die verbleibende Zeit bis zur oberen Schranke nicht unter eine Stunde fällt, wenn keine Kühlleistung vorhanden ist. Diese Zeit kann nach der Gleichung (4.2) abgeschätz werden.

$$\tau_{krit}(i) = \frac{m \cdot c \cdot (t_{max} - t(i))}{Q_{v_{ln}}(i)} \tag{4.2}$$

$$Q_{v_{ln}}(i) = \begin{cases} 1h \cdot \overline{\dot{Q}}_{Tr_{ln}}(i), & \text{außerhalb der Öffnungszeiten} \\ 1h \cdot (\overline{\dot{Q}}_{Tr_{ln}}(i) + \dot{Q}_{mehr}(i)), & \text{innerhalb der Öffnungszeiten} \end{cases}$$

$Q_{v_{ln}}(i)$ Verlustkältemenge zur Stunde i in kJ

$\tau_{krit}(i)$ Zeit bis zur kritischen Temperatur zur Stunde i in h

m Substanzmasse zur Aufnahme der Wärmeenergie in kg

c Spezifische Wärmekapazität der Substanzmasse in $\frac{\text{kJ}}{\text{kg}\cdot\text{K}}$

t_{max} obere Temperaturgrenze in °C

$t(i)$ Temperatur zur Stunde i in °C

$\overline{\dot{Q}}_{Tr_{ln}}(i)$ logarithmierte Mittelwert der Transmissionswärmeleistung zum aktuellen Zeitpunkt in kW

$\dot{Q}_{mehr}(i)$ Tagesmehrbedarf zum aktuellen Zeitpunkt in kW

Die Differenz zwischen der Transmissionswärmeleistung für den aktuellen Zeitpunkt wenn die Kühlraumtemperatur minimal wäre, und der Transmissionswärmeleistung zum aktuellen Zeitpunkt geteilt durch den Logarithmus aus der Division der beiden Leistungen, ergibt den logarithmierten Mittelwert der Transmissionswärmeleistung.

$$\overline{\dot{Q}}_{Tr_{ln}}(i) = \frac{\dot{Q}_{Tr_{max}}(i) - \dot{Q}_{Tr}(i)}{\log\left(\frac{\dot{Q}_{Tr_{max}}(i)}{\dot{Q}_{Tr}(i)}\right)} \tag{4.3}$$

$\overline{\dot{Q}}_{Tr_{ln}}(i)$ logarithmierte Mittelwert der Transmissionswärmeleistung zum aktuellen Zeitpunkt in kW

$\dot{Q}_{Tr_{max}}(i)$ Transmissionswärmeleistung für den aktuellen Zeitpunkt, wenn die Kühlraumtemperatur minimal wäre in kW

$\dot{Q}_{Tr}(i)$ Transmissionswärmeleistung für den aktuellen Zeitpunkt kW

$$\dot{Q}_{Tr_{max}}(i) = A \cdot k \cdot (t_{max} - t(i)) \tag{4.4}$$

$\dot{Q}_{Tr_{max}}(i)$ Transmissionswärmeleistung für den aktuellen Zeitpunkt, wenn die Kühlraumtemperatur minimal wäre in kW

A Fläche in m^2

k Wärmedurchgangskoeffiziente in $\frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}}$

t_{max} obere Temperaturgrenze in °C

$t(i)$ Temperatur zur Stunde i in °C

$$\dot{Q}_{Tr}(i) = A \cdot k \cdot (t_{amb}(i) - t(i)) \tag{4.5}$$

$\dot{Q}_{Tr}(i)$ aktuelle Transmissionswärmeleistung in kW

A Fläche in m^2

k Wärmedurchgangskoeffiziente in $\frac{\mathrm{W}}{\mathrm{m}^2 \cdot \mathrm{K}}$

$t_{amb}(i)$ aktuelle Umgebungstemperatur in °C

$t(i)$ aktuelle Kühlraumtemperatur in °C

Um den Temperaturausgleich in den Lebensmitteln im Algorithmus zu berücksichtigen, werden deshalb die zu- und abgeführten Wärmeenergiemengen bei der Berechnung der Temperatur für jeden Zeitschritt stets mit dem Faktor 0,8 multipliziert. Die Gleichung zur stündlichen Berechnung der aktuellen Temperatur ist damit:

$$t(i+1) = 0.8 \cdot \frac{Q_v - Q_{ab}}{m \cdot c} h + t(i) \tag{4.6}$$

t Temperatur in °C

Q_v eindringende Verlustwärmemenge in kJ

Q_{ab} abführende Wärmemenge in kJ

die aktuelle Gleichung gehört nicht dahin! oder?

wobei Q_v die aktuell eindringende Wärmeenergie und Q_{ab} die abgeführte Wärmeenergie ist.

In der Abbildung 4.4 auf der Seite 25 wird in einem Klassendiagramm die Beziehung zwischen dem bestehenden Modell und der Klasse **Cooling_strategy** dargestellt. Eine Klasse **Cooling_strategy** steht in der direkten Verbindung mit der Klasse **Refrigerator**, für die sie die zur Kühlung erforderliche Leistung berechnet. Ein Objekt der Klasse **Cooling_strategy** kann mit einem bis mehreren Objekten der Klasse **Refrigerator** in Verbindung stehen.

In der Abbildung 4.5 auf der Seite 27 ist der Programmstrukturplan des Kooperationskonzeptes dargestellt. Der Algorithmus zeigt den Ablauf beim "Auffühlen" der Kühlstellen für genau einen Zeitschritt der Simulation. Die Umsetzung des Algorithmus erfolgt in der Funktion **strategy_calculator()**.

strategy_calculator() Es gibt fünf Rückgabewerte dieser Funktion. Der erste Rückgabewert ist ein $n \times 11$-Array, wobei n für die Anzahl der im gesamten Netz vorhandenen Kühlzellen steht. In den Spalten des Arrays werden alle Informationen gespeichert, die durch Berechnungen in der Funktion entstanden sind und die Information zur eindeutigen Zuordnung dieser Ergebnisse zu der jeweiligen Kälteeinheit. ergibt sich dadurch, dass

das mache ich noch heute

Der Auffühlvorgang findet zwischen den stündlichen Temperaturberechnungen statt. Diese Handlung wird schrittweise wiederholt, um das gleichmäßige Verteilung der Leistung auf alle Anlagen zu ermöglichnen. Zur Lösung dieser Aufgabe im Programm ist eine **while**-Schleife mit einer Abbruchbedingung geeignet. Innerhalb der **while**-Schleife wird der Algorithmus Abbildung 4.5 umgesetzt.

Im ersten Schritt des Algorithmus findet eine Abfrage statt, ob Leistung zum Kühlen der Anlagen zur Verfügung steht. Dieser Berechnung liegt die Gleichung (4.1) auf der Seite 21 zugrunde, die in der Funktion **power_for_load_net()** implementiert ist.

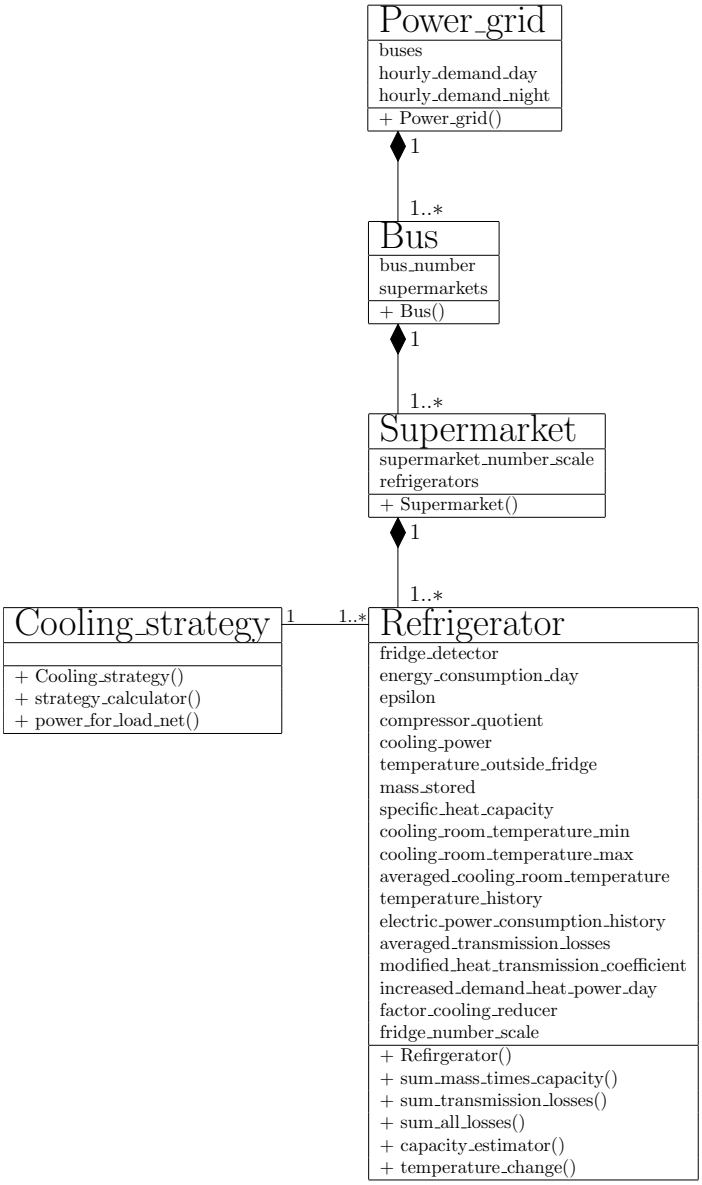

Abbildung 4.4: Klassendiagramm Kooperationskonzept

Fall 1 Die Menge der Leistung ist nicht größer Null. Es erfolgt die nächste Abfrage nach den Kühlstellen, die eine Stunde ohne Leistung aus dem Netz nicht überstehen können, ohne kritische Temperatur zu überschreiten. Diese Anlagen werden mit der zusätzlichen, nicht angemeldeten Leistung aus dem Netz geladen, sodass sie genau eine Stunde überstehen, ohne die obere Temperaturgrenze zu verletzen. Der Vorgang endet anschließend, und es kann zum nächsten Zeitschritt der Simulation übergegangen werden.

Sind keine solche Kühlstellen vorhanden, endet der Vorgang und es kann zum nächsten Zeitschritt der Simulation übergegangen werden.

Fall 2 Die Menge der Leistung ist größer Null. In diesem Fall werden alle

Kühlstellen gefunden, deren Kapazitätnoch nicht ausgeschöpft ist und deren Zeit bis zum erreichen der kritischen Temperatur am kleinsten ist. Anschließend wird überprüft, ob eine Wärmeenergie abgeführt werden kann, die genau $\frac{1}{60} \cdot {Q_v}^2$ entspricht, ohne als Folge die Überschreitung der minimalen Temperatur zu verursachen. Ist das der Fall, wird die Kapazität dieser Kühlstelle gleich Null gesetzt, damit diese Kühlstelle beim nächsten Durchgang der **while**-Schleife nicht beachtet wird.

was ist das?

Ist das Abführen der Wärmeenergiemenge ohne Verletzung der Temperaturgrenzen möglich, so wird anschließend die Kapazität, die zur Verfügung stehende Gesamtleistung und die Leistung, die zur Kühlung dieser Kühltruhe bereitgestellt wird, um diese Leistungsmenge ergänzt. Die Zeit τ_{krit} wird um einen Zeitschritt ergänzt. Der Zyklus der **while**-Schleife setzt sich fort.

$$Q_{Kap} = Q_{max} - \frac{1}{60} Q_v$$

$$P_{load} = P_{load} - \frac{1}{60} \frac{Q_v}{3.6 \cdot 10^6 \cdot \varepsilon}$$

$$\tau_{krit} = \tau_{krit} + \frac{1}{60}$$

$$Q_{cooling} = Q_{cooling} + \frac{1}{60} Q_v$$

Q_{Kap}	Wärmeenergiemenge die zusätzlich abgeführt werden kann, ohne dass die minimale Temperaturgrenze verletzt wird in kJ
Q_{max}	maximal abzuführende Wärmeenergiemenge in kJ
Q_v	eindringende Verlustwärmemenge in kJ
P_{load}	vorhandene Ladeleistung in MW
τ_{krit}	Zeit bis zur kritischen Temperatur in h
$Q_{cooling}$	Wärmeenergiemenge, die der Kühlzelle zur Kühlung zugewiesen wird in kJ

$$Q_{max} = m \cdot c \cdot (t(i) - t_{min}) + Q_v \tag{4.7}$$

Q_{max}	maximal abzuführende Wärmeenergiemenge in kJ
m	Substanzmasse zur Aufnahme der Wärmeenergie in kg
c	Spezifische Wärmekapazität der Substanzmasse in $\frac{\text{kJ}}{\text{kg·K}}$
Q_v	Verlustkältemenge in kJ
t_{min}	obere Temperaturgrenze in ° C
$t(i)$	Temperatur zur Stunde i in ° C

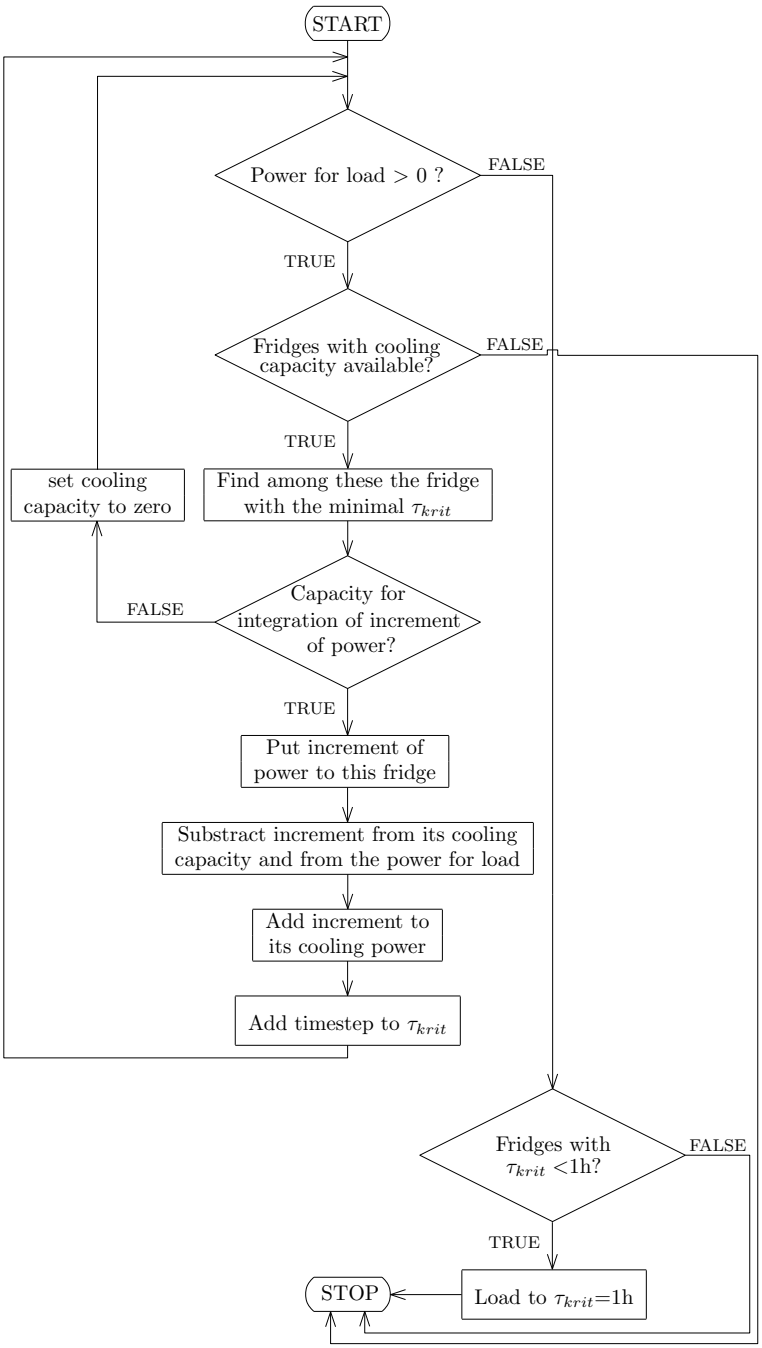

Abbildung 4.5: Flussdiagramm

Der genaue Ablauf des Informationsflusses, der zwischen den Objekten im Verlauf der Simulation stattfindet, wird im Sequenzdiagramm in der Abbildung 4.6 auf der Seite 28 grafisch verdeutlicht.

[2] Q_v ist die eindringende Verlustwärmemenge in kJ.

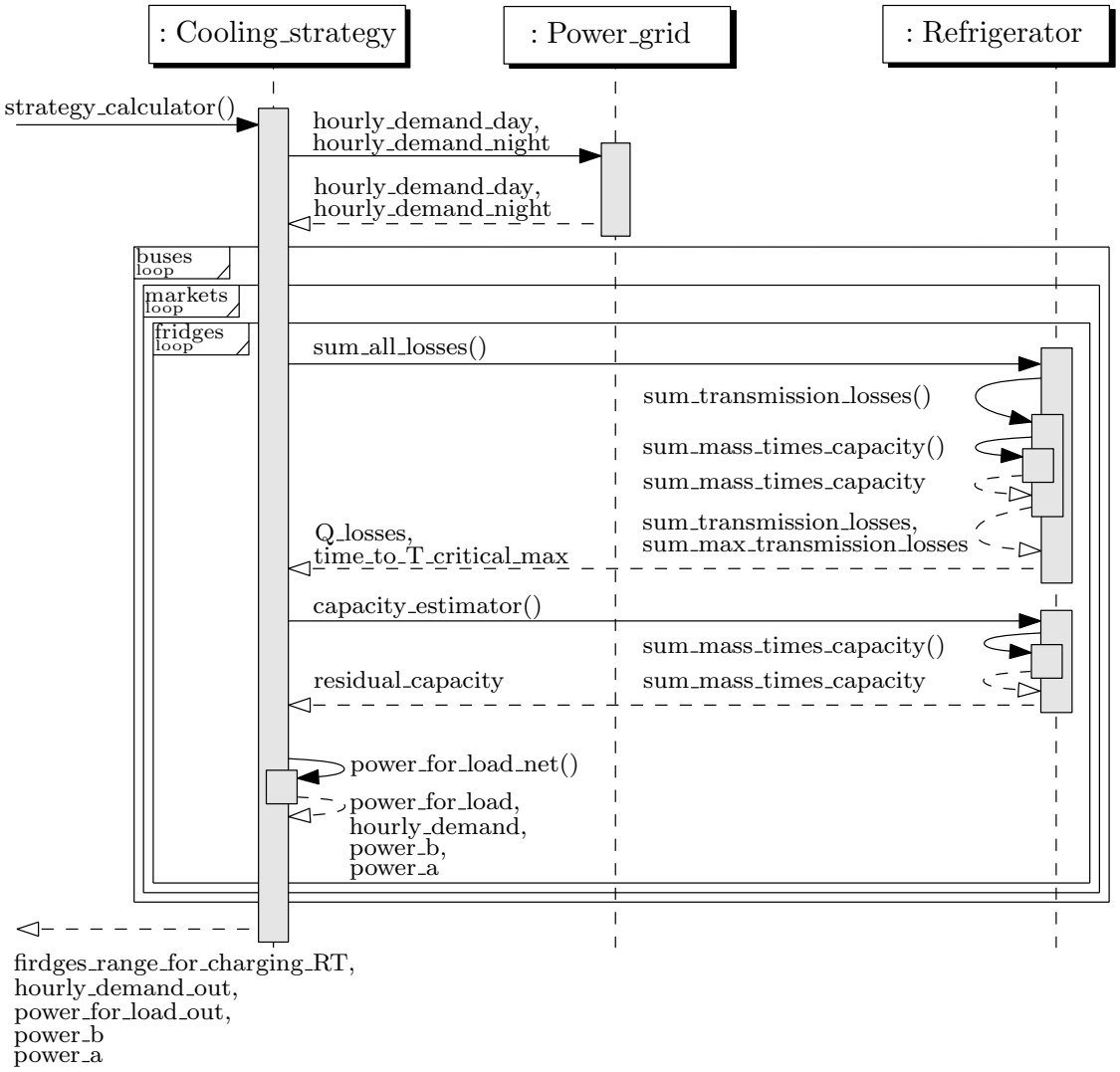

Abbildung 4.6: Sequenzdiagramm Kooperationsstrategie

4.2 Handhabung des Programms

In diesem Abschnitt wird beschrieben, we

4.2.1 Systemanforderungen

Zur Benutzung des Programms muss aufgrund der gewählten Programmiersprache und der objektorientierten Programmierwiese folgendes gelten:

- Im Rechner muss die Software MATLAB® der Version 5.0 oder höher[3] eingerichtet sein.

Die Anforderungen an die Hardware sind durch die eingesetzte MATLAB® - Version bestimmt.

[3] MATLAB® verfügbar über The MathWorks, Inc. (http://www.mathworks.com).

4.2.2 Installation

Das Programm richtet sich in seiner Installation und der Benutzung nach dem allgemein gültigen Gebrauch der MATLAB® M-files[4].

4.2.3 Aufruf der Simulation

Vorbereitung der Input-Information

Zum Starten des Programms werden folgende Konfigurationsdateien[5] benötigt:

- config_grid.m
- config_supermarkets.m
- config_fridges.m

config_grid.m

Es ist wichtig, dass die topologische Eigenschaft des Netzes im Programm berücksichtigt werden. Damit das Programm mit den für die Simulation notwendigen Informationen, die das Energienetz beschreiben, versorgt wird, ist die Form, die in MATLAB®-Code 4.1 vorgestellt wird, für die **config_grid.m**-Datei zwingend:

MATLAB®-Code 4.1: config_grid.m

```
%%  Bus,  Supermarkets,   Number of Supermarkets
configuration_grid = {...
        1 , {Aldi        ,        1500;      ...
             Netto       ,         500};     ...
        2 , {0           ,           0};     ...
        3 , {0           ,           0};     ...
        4 , {0           ,           0}      ...
       };
```

In der **config_grid.m**-Datei wird ein $n \times 2$-Cell Array ($n \in \mathbb{Z}_0^+$) definiert, der die Information über die Topologie des Netzes sowie die Verteilung der Kältelasten im Netz beinhaltet. Ein Cell Array ist ein Speicherobjekt, der verschiedene Datentypen unterschiedlicher Größe aufnehmen kann [1, Teil 2, Seite 15]. Ein Cell Array wird mit dem Befehl $c = cell(\ldots)$ oder mit Hilfe von geschweiften Klammern $c = \{\ldots\}$, wie in diesem Fall, erzeugt. Jede Zeile des Cell Arrays **configuration_grid**, der in der **config_grid.m**-Datei definiert werden muss, steht für ein Knotenpunkt im Netz. In der ersten Spalte wird die Nummer des Knotens

[4] Ausführliche Information dazu findet man z.B. im [1].

[5] Bei der Simulation wurden die Konfigurationsdateien verwendet, die im MATLAB®-Code A.1, MATLAB®-Code A.2 und MATLAB®-Code A.3 dargestellt sind.

bzw. Busses gespeichert. In die zweite Spalte werden die Arten der Kältelasten und deren Anzahl am jeweiligen Knoten festgelegt. Es ist wiederum ein $m \times 2$-Cell Array ($m \in \mathbb{Z}_0^+$). Jede Zeile dieses Cell Arrays ist für eine eigene Art Supermarktkette reserviert. In die erste Spalte kommt der Name der Supermarktkette deren Eigenschaften in der **config_supermarkets.m**-Datei (vgl. MATLAB®-Code 4.2) gespeichert sind. In der zweiten Spalte wird die Anzahl der Supermärkte einer Kette festgelegt, die an dem bestimmten Knoten simuliert werden soll.

Um Fehler auf dieser Ebene zu vermeiden, muss die **config_grid.m**-Datei unbedingt die oben vorgestellte Form beibehalten.

config_supermarkets.m

MATLAB®-Code 4.2: config_supermarkets.m

```
%%       Kind of fridge     Anzahl
Aldi = {...
        { NK_KT_S       ,    1 };      ...
        { NK_KR_V       ,    2 };      ...
        { TK_TKT_S      ,    1 }       ...
       };
Netto = {...
        { NK_KT_S       ,    5 };      ...
       };
```

Die Kältelast in einem Supermarkt bilden die einzelnen je nach Einsatzzweck speziell dafür konstruierten Kühleinheiten. Die Anzahl, die technischen Eigenschaften und die Betriebsweise dieser Kältemaschinen können in der Realität Unterschiede aufweisen. In der Konfigurationsdatei **config_supermarkets.m** (vgl. MATLAB®-Code 4.2) wird festgelegt, wie die einzelnen Supermarktketten aus verschiedenen Kälteeinheiten zusammengesetzt sind. Die Gruppierung der einzelnen Kühleinheiten zu einem Modellsupermarkt erfolgt in der **config_supermarkets.m**-Datei durch Definition eines oder mehrerer $i \times 2$-Cell ($i \in \mathbb{Z}_0^+$) Arrays, die jeweils ein Supermarkt abbilden. Jede Zeile eines solchen Cell Arrays ist für je eine Art Kühleinheit reserviert. In die erste Spalte kommt der Name der Kühlzelle, deren Eigenschaften in der **config_fridges.m**-Datei (vgl. MATLAB®-Code 4.3 auf der Seite 31) gespeichert sind. In der zweiten Spalte wird die Anzahl für diese Kühleinheiten, die in dem Supermarkt betrieben werden, festgelegt.

config_fridges.m

Die Konfigurationsdatei **config_fridges.m** kann als eine Datenbank für Kälteeinheiten aufgefasst werden. In dieser Datenbank werden Informationen nach einem bestimmten Muster gruppiert und gespeichert, sodass jede Gruppe eine bestimmte Kühleinheit abbildet. Im Programm kann aus je einem dieser physikalisch

fundierten Modellen ein Kühleinheit-Objekt erzeugt werden.

Am Beispiel des Modells einer steckerfertigen Normalkühltruhe, abgekürzt NK_KT_S (vgl. MATLAB®-Code 4.3 auf der Seite 31), wird im folgenden die Eingabeform eines solchen Datenbankeintrages in der Datei **config_fridges.m** erläutert. Zur Beschreibung einer Kühleinheit ist erforderlich, die Annahmen in einen 1×14-Cell Array zu erfassen. Die Größe und die Art des Arrays ist vorgegeben durch die Auslegung des Modells zur Anzahl und dem Typus der modellbeschreibenden Annahmen. Die Struktur eines Cell Arrays ermöglicht neben der zusammenhängenden Speicherung verschiedener Datentypen einen durch MATLAB® darauf direkten Zugriff.

MATLAB®-Code 4.3: config_fridges.m

```
 % fridge configuration parameters
 NK_KT_S = { ...
1          1, ... % 1 if plugin module or 2 if combine fridge
2          4.7e3, ... % energy consumption per day Wh/24h
3          2, ... % epsilon power quotient
4          0.66, ... % compressor quotient
5          0 ... % installed cooling power in W
6          [16.4 6.7], ... % area_wall
7          [0.38 0.38], ... % heat_transmission_coefficient
8          [19 15], ... % temperature_outside
9          [200 200], ... % masse_stored
10         [2.3 3.52], ... % specific_mass_capacity
11         -6, ... % temperature_min in Celsius
12         2, ... % temperature_max in Celsius
13         1, ... % averaged cooling room temperature in Celsius
14         0, ... % refrigerating capacity
         };
```

Spalte eins [6] Die Unterscheidung in steckerfertige Kälteanlagen und Kälteverbundanlagen im Programm ist notwendig, da bei der Berechnung der Kälteverluste unterschiedliche Verfahren zu Grunde gelegt werden. Diese Differenzierung erfolgt durch die Zuweisung einer Eins für die steckerfertige Einheit und einer Zwei für eine Einheit an Verbundanlage in der ersten Spalte im Array.

Spalte zwei Die druchschnittliche elektrische Energieaufnahme ist eine Kennzahl in Watt pro 24 Stunden, die durch die Hersteller mit Hilfe eines genormten Verfahren ermittelt wird und in den technischen Blättern angegeben werden muss.

Wo wird das Verwendet? Hier schreiben

Spalte drei In diese Spalte wird die Leistungszahl eingetragen.

[6] Aus Gründen der Übersicht sind im MATLAB®-Code 4.3 auf der Seite 31 der Zeilenvektor NK_KT_S als Spaltenvektor dargestellt. Es wird jedoch weiterhin auf Spalten verwiesen, da er in MATLAB® als Zeilenvektor implementiert ist.

Spalte vier Der Anteil des Verdichters an dem Tagesenergieverbrauch der Kühleinheiten wird Leistungszahl genannt. Aus der Gleichung (2.11) im Kapitel 2 geht hervor, wie diese Kehnzahl in die Berechnung eingeht. Die Liestungszahl wird in der Spalte vier festgehalten.

Spalte fünf Werden anschlussfertige Kälteaggregate zur Kühlung von Räumen verwendet, so wird die installierte Kälteleistung in Kilowatt in die Spalte fünf anstatt einer Null eingetragen.

Spalte sechs Ein Teil der Wärmeverluste ist auf die Transmissionsverluste durch die Wände zurückzuführen. Die Anzahl der Wände muss für die Rechnung berücksichtigt werden, Sie wird in die sechste Spalte eingetragen.

Spalte sieben Für die Transmissionsverlustberechnung sind unter anderem die Flächengrößen und die jeweiligen Wärmedurchgangskoeffizienten der einzelnen Wände maßgebend[7]. Für je eine Flächengröße in Quadratmeter ist in der Spalte sieben des NK_KT_S-Arrays ein Spaltenplatz im $1 \times w$-Zeilenarray reserviert. Die w steht für die Anzahl der Wände.

Spalte acht In der achten Spalte werden in einem weiteren $1 \times w$-Zeilenarray die mit den Flächen korrespondiere Wärmedurchgangskoeffiziente[7] in gleicher Reihenfolge gespeichert.

Spalte neun Die Tepmeratur außerhalb der Kühleinheit bestimmt[7] die Größe der Transmissionsverluste. In der neunten Spalte werden nach dem gleichen Prinzip und in der gleichen Form wie im vorhergegangenen Eintrag die Außentemperatur für jede Wand eingetragen.

Spalte zehn Die Abhängigkeit der Temperaturänderung von der in einer Kühleinheit deponierten Masse an Waren und deren spezifische Wärmekapazität ist aus der Gleichung (2.1) im Kapitel 2 ersichtlich. In der Realität werden in einem Kühlschrank gewöhnlich mehrere verschiedene Lebensmittel gekühlt. Möchte man gemischte Beladung simulieren, so sind die Massen in der Spalte zehn im NK_KT_S in der gleichen Art und Weise wie in der Spalte sieben acht oder neun zu einzutragen. Der Eintrag erfolgt in Kilogramm.

Spalte elf Zu jeder in der Spalte zehn eingetragener Massezahl muss in der Spalte neun nach dem Beispiel der Spalte acht die spezifische Wärmekapazität eingetragen werden.

Spalte zwölf In der Realität fällt die Spanne für Variation die Innentemperatur in einem Lebensmittelkühlschrank eher gering aus, da die Temperatur auf einem bestimmten Niveau gehalten werden muss, damit die Lebensmittel maximal lange frisch bleiben können. Möchte man die Kälte im Lebensmittel speichern, darf die Lebensmitteltemperatur einen bestimmten Bereich nicht Verlassen.

[7] Der mathematische Zusammenhang wird im Kapitel 2 in der Gleichung (2.3) deutlich gemacht.

In der Spalte zwölf im NK_KT_S wird die untere Temperaturgrenze in Grad Celsius festgelegt.

Spalte elf In der Spalte elf wird die obere Temperaturgrenze in Grad Celsius festgelegt.

Spalte zwölf Die Bedeutung[8] Ausführlich begründet wird das in [10]. der mittleren Kühlraumtemperatur ist kurz im Kapitel 2 angeschnitten. Notiert wird diese Kennzahl in der Spalte zwölf im NK_KT_S eingetragen.

Klären was das ist.

Spalte dreizehn Keine Ahnung

Die main.m Datei

4.2.3.1 Deklamation der berechneten Daten

Die inhaltliche Auswertung der Simulation ist dem Benutzer vorbehalten. Darum müssen die Ergebnisse der Simulation in einer anwenderfreundlichen Form zugänglich gemacht werden. Durch sinnvolle graphische Darstellung der untersuchten Größen kann zum Beispiel die Überprüfung, die Veranschaulichung oder die Bestimmung der funktionalen Abhängigkeit dieser durchgeführt werden.

Das ist noch nicht richtig im Programm implementiert. Das muss DU NOCH MACHEN!

Graphische Ausgabe

Wichtig!!! hier wird nicht der Quellcode erklärt, sondern die Darstellung der Ausgabe.

Funktion mit für die graphische Ausgabe schreiben

4.2.3.2 Tabellarische Ausgabe im Command-Window und Exel

Funktion für tabellarische Ausgabe schreiben

[8] caro

Anhang A

Anhang

A.1 Anhang Example section

This for other bibtex stye file: only [10] one author [1] and many authors [14].

MATLAB®-Code A.1: config_grid.m

```
%% The conf.m file have to be in this format for make the program run.
%%         Bus,  Supermarkets,           number of Supermarkets
config_grid_ = {...
        1 , {Aldi         ,        1500};     ...
        2 , {0            ,           0};     ...
        3 , {0            ,           0};     ...
        4 , {0            ,           0}      ...
          };
```

MATLAB®-Code A.2: config_supermarkets.m

```
%
Aldi = {...
        {NK_KT_S        ,           1};     ...
        {NK_KR_V        ,           1};     ...
        {TK_TKT_S       ,           1};     ...
        {NK_KZ          ,           1};     ...
        {TK_KZ          ,           1}      ...
       };

% Aldi = {...
%          { NK_KT_S ,   1}     ...
%        };
```

MATLAB®-Code A.3: config_fridges.m

```
NK_KT_S = { ...
    1, ... % kind of fridge: one plug in or two combine fridge
    1.88e4, ... % energy consumption per day Wh/24h
    2, ... % epsilon
    0.66, ... % compressor quotient
    0, ... % cooling power in W
    [16.4 6.7], ... % area_wall
    [0.38 0.38], ... % heat_transmission_coefficient
    [19 15], ... % temperature_outside
```

```
    [200 200]... % masse_stored
    [2.3 3.52], ... % specific_heat_capacity for each mass
    0, ... % temperature_min in C
    4, ... % temperature_max in C
    2, ... % averaged cooling room temperature in C
    0, ... % refrigerating capacity
    0.75 ... % factor for reducing of cooling demand
    };

TK_TKT_S = { ...
    1, ... % kind of fridge: one plug in or two combine fridge
    8.88e4, ... % energy consumption per day Wh/24h
    1.5, ... % epsilon
    0.66, ... % compressor quotient
    0, ... % cooling power in W
    [36.9 16.8] ... % area_wall
    [0.38 0.38], ... % heat_transmission_coefficient
    [19 15], ... % temperature_outside
    [480 840 360 756]... % masse_stored
    [1.76 1.76 1.38 1.91], ... % specific_mass_capacity
    -26, ... % temperature_min in C
    -18, ... % temperature_max in C
    -22, ... % averaged cooling room temperature in C
    0, ... % refrigerating capacity
    0.75}; % factor for reducing of cooling demand

NK_KR_V = { ...
    2, ... % kind of fridge: one plug in or two combine fridge
    0, ... % energy consumption per day Wh/24h
    2, ... % epsilon
    0, ... % compressor quotient
    28.3e3, ... % cooling power in W
    [40.2 68.8 24.3], ... % area_wall
    [1.53 0.62 0.38], ... % heat_transmission_coefficient
    [19 19 15], ... % temperature_outside
    [1210 1050 2520], ... % masse_stored
    [2.51 2.3 3.85], ... % specific_mass_capacity
    2, ... % temperature_min in C
    8, ... % temperature_max in C
    5, ... % averaged cooling room temperature in C
    0, ... % refrigerating capacity
    0.75}; % factor for reducing of cooling demand

NK_KZ = { ...
    2, ... % kind of fridge: one plug in or two combine fridge
    0, ... % energy consumption per day Wh/24h
    2, ... % epsilon
    0, ... % compressor quotient
    830, ... % cooling power in W
    [30.7], ... % area_wall
    [0.26], ... % heat_transmission_coefficient
    [15], ... % temperature_outside
    [1440 2304], ... % masse_stored
```

```
    [2.51 3.85], ... % specific_mass_capacity
    1, ... % temperature_min in C
    5, ... % temperature_max in C
    3,  ... % averaged cooling room temperature in C
    0, ... % refrigerating capacity
    1}; % factor for reducing of cooling demand

TK_KZ = { ...
    2, ... % kind of fridge: one plug in or two combine fridge
    0, ... % energy consumption per day Wh/24h
    1.5, ... % epsilon
    0, ... % compressor quotient
    1060 ... % cooling power in W
    [30.7], ... % area_wall
    [0.21], ... % heat_transmission_coefficient
    [15], ... % temperature_outside
    [1522 1384], ... % masse_stored
    [1.76 1.76], ... % specific_mass_capacity
    -26, ... % temperature_min in C
    -18, ... % temperature_max in C
    -22, ... % averaged cooling room temperature in C
    0, ... % refrigerating capacity
    1}; % factor for reducing of cooling demand
```

MATLAB®-Code A.4: runCooling.m

```
function [  load_output, ...
        grid, ...
        multiple_dimension_all_data_matrix, ...
        control_power_WO, ...
        control_power_W] = runCooling(config_grid)
    %% this function estimates:
    % a) r = load_matrix for power flow simulator
    % b) grid_system = supermarket system whit all estimated values and
    % parameters
    % c) b = matrix whit provided and integrated electric power
    % here are indices of time the simulation should run
    number_steps = 24;
    number_days = 29;
    % object grid system class Power_grid initialised
    grid = Power_grid(config_grid, number_steps, number_days);
    % object strategy class Cooling_strategy initialised
    strategy = Cooling_strategy;

    %%--------------load Data-----------------%%
    load 'gen_output_RT'
    load 'gen_output_DA'
    % this is the real power that wind power can supply
    gen_output_real_T = gen_output_RT(4:7,:,1:number_days);
    gen_output_day_A = gen_output_DA(4:7,:,1:number_days);
    promised_gen_output_DA_gross = ...
       sum(gen_output_DA(4:7,:,1:number_days) * 0.0015);
```

```
%%-------------Data loaded-----------------%%

load_output = zeros(size(gen_output_RT));
multiple_dimension_all_data_matrix = ...
   zeros([size(gen_output_RT) 7]);

%% loop run down the "w" days of year
for w = 1 : number_days
    %% loop run down the "l" time steps of the day
    for l = 1 : number_steps
        % invoke function every time step
        [   fridges_range_for_load_RT, ...
            hourly_demand_out, ...
            power_for_load_out, ...
            power_b, ...
            power_a] = ...
                strategy.strategy_calculator( ...
                grid.buses, ...
                grid.hourly_demand_day, ...
                grid.hourly_demand_night, ...
                w, ...
                l, ...
                gen_output_real_T(:,l,w), ...
                gen_output_day_A(:,l,w), ...
                promised_gen_output_DA_gross(:,l,w), ...
                number_steps, ...
                number_days);

        for m = 1:length(fridges_range_for_load_RT(:,1))

    b = fridges_range_for_load_RT(m,2); % number_bus
    s = fridges_range_for_load_RT(m,3); % number_steps
    r = fridges_range_for_load_RT(m,4); % number_fridge

            % TEMPERATURE CALCULATION
            grid.buses(b).supermarkets(s).refrigerators(r). ...
        temperature_change( ...
        number_steps, ...
        l, ... % count_step
        w, ... % count_day
        fridges_range_for_load_RT(m,5), ... % cooling
        fridges_range_for_load_RT(m,6)); % losses

            % THE RETURN LOAD OUTPUT MATRIX of the WILL BE WRITTEN
            load_output(b,l,w) = load_output(b,l,w) + ...
        fridges_range_for_load_RT(m,10);

            % THE RETURN ELECTRIC POWER INTEGRATION MATRIX
            % overall electric power consumption
    multiple_dimension_all_data_matrix(b,l,w,1) = ...
        hourly_demand_out; %fridges_range_for_load_RT(m,10);

    % overall electric power integrated
```

```
        multiple_dimension_all_data_matrix(b,l,w,2) = ...
            multiple_dimension_all_data_matrix(b,l,w,2) + ...
            fridges_range_for_load_RT(m,11);


                % overall power for load
        multiple_dimension_all_data_matrix(b,l,w,3) = ...
            power_for_load_out(b);

                % excess wind power
        multiple_dimension_all_data_matrix(b,l,w,4) = ...
            abs(gen_output_real_T(b,l,w) - power_b(b));

                % overall power
        multiple_dimension_all_data_matrix(b,l,w,5) = ...
            multiple_dimension_all_data_matrix(b,l,w,5) + ...
            fridges_range_for_load_RT(m,10);

        multiple_dimension_all_data_matrix(b,l,w,6) = ...
            multiple_dimension_all_data_matrix(b,l,w,5) - ...
            multiple_dimension_all_data_matrix(b,l,w,2);

        multiple_dimension_all_data_matrix(b,l,w,7) = power_a(b);
            end

            load_output(b,l,w) = load_output(b,l,w) - hourly_demand_out;

        end
    end
    %%
    control_power_WO = sum(sum(sum(abs( ...
       gen_output_real_T(:,1:number_steps,:) - ...
       gen_output_day_A(:,1:number_steps,:)))));

    control_power_W = ...
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,4))))- ...
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,2)))) + ...
    (sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,1)))) - ...
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,7))))) + ...
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,6))));

    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,2))))
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,6))))
    sum(sum(sum(multiple_dimension_all_data_matrix(:,:,:,7))))

end
```

MATLAB® -Code A.5: Cooling_strategy.m

```
classdef Cooling_strategy < handle
 % NEW_COOLING_STRATEGY Summary of this class goes here
 %   Detailed explanation goes here
```

```
properties
end

methods

 function obj = Cooling_strategy
 end % function constructor

 function [ fridges_range_for_charging_RT, ...
       hourly_demand_out, ...
       power_for_load_out, ...
       power_b, ...
       power_a] = ...
           strategy_calculator( obj, ...
              buses, ...
              hourly_demand_day, ...
              hourly_demand_night, ...
              count_number_days, ...
              count_number_steps, ...
              gen_out_day_T, ...
              gen_out_day_A, ...
              promised_gen_output_DA_gros, ...
              number_steps, ...
              number_days)

 % This function gives a matrix which contains rang of
 % refrigerators refer to size up time to the critical temperature
 cooling = 0; %
 count_bus = 0; % initialisation runner
 time_step = 1/60; % incrementally charging
 electric_power_consumption = 0;
 wind_power_integrated = 0; % initialisation runner
 fridges_range_for_charging_RT = zeros(1,11);

 for b = 1:length(buses) % each bus
    count_bus = count_bus + 1; % update runner
    count_supermarket = 0; % initialisation runner
    for s = 1 : length(buses(b).supermarkets) %each supermarket
        count_supermarket = count_supermarket + 1;
        count_refrigerator = 0;
        for r = 1:length([buses(b).supermarkets(s).refrigerators])

      % before the estimation of temperature for the next step
      % can be started, the decision for the power, which
      % will be adjudged for each fridge, has be affected
      [Q_losses, time_to_T_critical_max] = ...
         buses(b).supermarkets(s).refrigerators(r). ...
              sum_all_losses(...
              number_steps, ...
              count_number_steps, ...
              count_number_days, ...
              number_days);
```

```
      % this function estimates the heat power the fridge can
      % maximum integrate for the moment
      residual_capacity = buses(b). ...
        supermarkets(s). ...
        refrigerators(r). ...
        capacity_estimator( ...
            count_number_steps, ...
            count_number_days, ...
            Q_losses);

            if sum (fridges_range_for_charging_RT) == 0 % securing

                count_refrigerator = count_refrigerator + 1;
                fridges_range_for_charging_RT = [...
                    time_to_T_critical_max ...
                    count_bus ...
                    count_supermarket ...
                    count_refrigerator ...
                    cooling ...
                    Q_losses ...
                    time_step ...
                    buses(b).supermarkets(s).refrigerators(r). ...
                    epsilon ...
                    residual_capacity ...
                    electric_power_consumption ...
                    wind_power_integrated ...
                    ];

            else
                count_refrigerator = count_refrigerator + 1;
                fridges_range_for_charging_RT = [...
                    fridges_range_for_charging_RT; ...
                    time_to_T_critical_max ...                  %1
                    count_bus ...                               %2
                    count_supermarket ...                       %3
                    count_refrigerator ...                      %4
                    cooling ...                                 %5
                    Q_losses ...                                %6
                    time_step ...                               %7
                    buses(b).supermarkets(s).refrigerators(r). ...
                    epsilon ...                             %8
                    residual_capacity ...                       %9
                    electric_power_consumption ...             %10
                    wind_power_integrated ...                  %11
                    ];
            end
        end
    end
end

% condition for load fridges
MODUS = 'load';
```

```
[ power_for_load, ...
  hourly_demand, ...
  power_b, ...
  power_a] = obj.power_for_load_net( ...
              count_number_steps, ...
              gen_out_day_T, ...
              gen_out_day_A, ...
              promised_gen_output_DA_gros, ...
              hourly_demand_day, ...
              hourly_demand_night);

% TODO muss weg
hourly_demand_out = hourly_demand;
power_for_load_out = power_for_load;
while strcmp('load', MODUS)

    table_to_restock = 0;

  if power_for_load > 0
    for c = find(fridges_range_for_charging_RT(:,9) > 0)
      if table_to_restock(1) == 0
      table_to_restock = [c, fridges_range_for_charging_RT(c,:)];
      else
      table_to_restock = [table_to_restock; c, ...
      fridges_range_for_charging_RT(c,:)];
      end
    end

    if length(table_to_restock) > 1

    [~,t] = min(table_to_restock(:,2));
    n = table_to_restock(t,1);

    % heat from cooling (power demand) for one more minute
    transitional_storage_power_for_one_more_minute = ...
    fridges_range_for_charging_RT(n,7) * ...
    fridges_range_for_charging_RT(n,6);
      % this request secures that the fridge do not overload
      if fridges_range_for_charging_RT(n,9) - ...
         (fridges_range_for_charging_RT(n,6) * time_step) < 0
                %
         fridges_range_for_charging_RT(n,9) = 0;
      else
         % save the time period that was integrated
         fridges_range_for_charging_RT(n, 1) = ...
         fridges_range_for_charging_RT(n, 1) + ...
         fridges_range_for_charging_RT(n, 7);

         % update residual_capacity
         fridges_range_for_charging_RT(n, 9) = ...
         fridges_range_for_charging_RT(n, 9) - ...
         time_step * fridges_range_for_charging_RT(n,6);
         % update power_for_load
```

```
        power_for_load = power_for_load - ...
        (transitional_storage_power_for_one_more_minute / ...
        fridges_range_for_charging_RT(n, 8) / 3.6e6) * ...
        buses(fridges_range_for_charging_RT(n, 2)). ...
        supermarkets(fridges_range_for_charging_RT(n, 3)). ...
        supermarket_number_scale * ...
        buses(fridges_range_for_charging_RT(n, 2)). ...
        supermarkets(fridges_range_for_charging_RT(n, 3)). ...
        refrigerators(fridges_range_for_charging_RT(n,4)). ...
        fridge_number_scale;
        % update cooling
        fridges_range_for_charging_RT(n, 5) = ...
        fridges_range_for_charging_RT(n,5) + ...
        time_step * fridges_range_for_charging_RT(n,6);
        % update of electric_power_consumption
        fridges_range_for_charging_RT(n,10) = ...
        (fridges_range_for_charging_RT(n,5) / ...
        fridges_range_for_charging_RT(n,8) / 3.6e6) * ...
        buses(fridges_range_for_charging_RT(n, 2)). ...
        supermarkets(fridges_range_for_charging_RT(n, 3)). ...
        supermarket_number_scale * ...
        buses(fridges_range_for_charging_RT(n, 2)). ...
        supermarkets(fridges_range_for_charging_RT(n, 3)). ...
        refrigerators(fridges_range_for_charging_RT(n,4)). ...
        fridge_number_scale;

        % integrate_power update
        fridges_range_for_charging_RT(n,11) = ...
          fridges_range_for_charging_RT(n,10);
    end
  else
     MODUS = 'do_not_load';
  end
 else
  for v = 1 : length(fridges_range_for_charging_RT(:,1))
      if fridges_range_for_charging_RT(v,1) < 1
         % estimates the rest of the time_crit to one hour
         s = 1 - fridges_range_for_charging_RT(v,1);
         % power cooling updated
         fridges_range_for_charging_RT(v,5) = ...
         fridges_range_for_charging_RT(v,5) + ...
         s * fridges_range_for_charging_RT(v,6);
         % update of electric_power_consumption
         fridges_range_for_charging_RT(v,10) = ...
         (fridges_range_for_charging_RT(v,5) / ...
         fridges_range_for_charging_RT(v,8) / 3.6e6) * ...
         buses(fridges_range_for_charging_RT(v, 2)). ...
         supermarkets(fridges_range_for_charging_RT(v, 3)). ...
         supermarket_number_scale * ...
         buses(fridges_range_for_charging_RT(v, 2)). ...
         supermarkets(fridges_range_for_charging_RT(v, 3)). ...
         refrigerators(fridges_range_for_charging_RT(v,4)). ...
         fridge_number_scale;
```

```
        else
            MODUS = 'do_not_load';
        end
    end
    MODUS = 'do_not_load';
   end
  end
 end

 % this function estimates power_for_load_nett
 function [power_for_load, hourly_demand, power_b, power_a] = ...
       power_for_load_net(obj, ...
               count_number_steps, ...
               gen_out_day_T, ...
               gen_out_day_A, ...
               promised_gen_output_DA_gros, ...
               hourly_demand_day, ...
               hourly_demand_night)

     if count_number_steps < 9 || count_number_steps > 20
         hourly_demand = hourly_demand_night;
     else
         hourly_demand = hourly_demand_day;
     end
     if promised_gen_output_DA_gros > hourly_demand
         power_a = hourly_demand;
         power_b = sum(gen_out_day_A) - hourly_demand;
     else
         power_a = promised_gen_output_DA_gros;
         power_b = sum(gen_out_day_A) - promised_gen_output_DA_gros;
     end
     if sum(gen_out_day_T) > sum(power_b)
         power_for_load = sum(gen_out_day_T) - sum(power_b) + ...
         hourly_demand - power_a;
     else
         power_for_load = hourly_demand - power_a;
     end

 end
 end
end
```

MATLAB®-Code A.6: Power_grid.m

```
classdef Power_grid < handle % handle call by reference
    %Power_grid Summary of this class goes here
    % Detailed explanation goes here
    properties
    buses %
    hourly_demand_day %
    hourly_demand_night %
    end
```

```
    methods
    % invoke constructor class Power_grid
        function obj = Power_grid( ...
                configuration_grid, ...
                number_steps, ...
                days ...
                )
    % in configuration_grid all data about which how many
    % supermarkets are connected on which bus invoce
    % constructor class Bus and save into
    % properties buses of object Power_grid
        for i = 1 : configuration_grid{end,1}
            obj.buses = [obj.buses ...
                Bus({configuration_grid{i,2}}, ...
                i, ...
                number_steps, ...
                days)];
        end
    obj.hourly_demand_day = 0;
    obj.hourly_demand_night = 0;
%       here hourly_demand_day and %_night calculation starts
        % observance of all supermarkets on each bus
            for b = 1 : length(obj.buses)

        % observance of all refrigerators of each supermarket
                for s = 1 : length(obj.buses(b).supermarkets)
            super = obj.buses(b).supermarkets(s); % substitution
                    for r = 1 : length(super.refrigerators)

            refr = super.refrigerators(r); % substitution

            obj.hourly_demand_day = obj.hourly_demand_day + ...
            (refr.increased_demand_heat_power_day ...
            / 3.6 + refr.averaged_transmission_losses) * ...
            refr.fridge_number_scale * ...
            super.supermarket_number_scale ...
            / refr.epsilon;

            obj.hourly_demand_night = obj.hourly_demand_night + ...
            refr.averaged_transmission_losses * ...
            refr.fridge_number_scale * ...
            super.supermarket_number_scale ...
            / refr.epsilon;
                    end
                end
            end
            obj.hourly_demand_day = obj.hourly_demand_day / 1e6;
            obj.hourly_demand_night = obj.hourly_demand_night / 1e6;
        end
    end
end
```

MATLAB®-Code A.7: Bus.m

```
classdef Bus < handle % have to be handle class becouse of the speed
 %%Bus Summary of this class goes here
 %   Detailed explanation goes here
 %%
 properties
     bus_number % the number of the bus
     supermarkets % all supermarkets which are connected into bus
 end % end properties
 %%
 methods
 % the constuctor of Bus class
     function obj = Bus( ...
                supermarkets_connected, ...
                bus_number, ...
                number_steps, ...
                days ...
            )

         obj.bus_number = bus_number; % save bus_number
         for n = 1 : length([supermarkets_connected{1}{:,2}])
             if supermarkets_connected{1}{n,2} ~= 0
                obj.supermarkets = [obj.supermarkets ...
                    Supermarket( ...
                    {supermarkets_connected{1}{n,1}}, ...
                    supermarkets_connected{1}{n,2}, ...
                    number_steps, ...
                    days)];
             else
                 obj.supermarkets = [obj.supermarkets []];
             end
         end
     end
     %%
  end % methods end
end
```

MATLAB®-Code A.8: Supermarket.m

```
classdef Supermarket < handle
 %SUPERMARKET Summary of this class goes here
 %   Detailed explanation goes here
 %%
 properties
     supermarket_number_scale
     refrigerators
 end
 %%
 methods
   function obj = Supermarket( ...
      fridges, ...
      supermarket_number_scale, ...
```

```
      number_steps, ...
      days)

       obj.supermarket_number_scale = supermarket_number_scale;
       for n = 1 : length(fridges{1})
           obj.refrigerators = [obj.refrigerators ...
               Refrigerator( ...
                 fridges{1}{n},  ...
                 number_steps, ...
                 days )];
       end % end for
   end % end function Supermarket
 end % end methods
end % end classdef
```

MATLAB®-Code A.9: Refrigerator.m

```
classdef Refrigerator < handle
 %% REFRIGERATOR class created from Juri Steblau 09.03.10
 %   this class is a part of an program calls SuperM which simulates an
 %   supermarket as a cooling energy storage
 %%
 properties
     fridge_detector % the identification number of refrigerator
     energy_consumption_day % the maxim on power refrigerator can use
     epsilon % the ...
     compressor_quotient
     cooling_power
     temperature_outside_fridge
     mass_stored % the mass of stored product
     specific_heat_capacity % for each mass
     cooling_room_temperature_min % the minimum on temperature
     cooling_room_temperature_max % the maximum on temperature
     averaged_cooling_room_temperature
     temperature_history % temperature into fridge
     electric_power_consumption_history % the power consumption real
     averaged_transmission_losses
     modified_heat_transmission_coefficient
     increased_demand_heat_power_day % rest_day_power
     factor_cooling_reducer
     fridge_number_scale % factor for scale number fridges
 end % properties end
 %%
 methods
  function obj = Refrigerator( fridge_config, ...
                               number_steps, ...
                               days ) % fridge constructor
      % save the object properties
      obj.fridge_detector = fridge_config{1,1}{1}; % kind of fridge
      obj.energy_consumption_day = fridge_config{1,1}{2}; % in Wh/h
      obj.epsilon = fridge_config{1,1}{3};
      obj.compressor_quotient = fridge_config{1,1}{4};
```

```
      obj.cooling_power =  fridge_config{1,1}{5};
      obj.temperature_outside_fridge = fridge_config{1,1}{8};
      obj.mass_stored = fridge_config{1,1}{9};
      obj.specific_heat_capacity =fridge_config{1,1}{10};
      obj.cooling_room_temperature_min = fridge_config{1,1}{11};
      obj.cooling_room_temperature_max = fridge_config{1,1}{12};
      obj.averaged_cooling_room_temperature = fridge_config{1,1}{13};
      obj.temperature_history = zeros(number_steps, days);
      obj.electric_power_consumption_history =  ...
          zeros(number_steps, days);
      obj.temperature_history(1,1) =  ...
          obj.averaged_cooling_room_temperature;
      obj.factor_cooling_reducer = fridge_config{1,1}{15};
      obj.fridge_number_scale = fridge_config{2};
      obj.modified_heat_transmission_coefficient = ...
          fridge_config{1,1}{6} .* fridge_config{1,1}{7}; % Watt/K end
      % estimation of averaged transmission losses
      obj.averaged_transmission_losses = sum( ...
          obj.modified_heat_transmission_coefficient .* ...
          ( obj.temperature_outside_fridge - ...
          obj.averaged_cooling_room_temperature ));

      % this function estimates the rest of the power
      if obj.fridge_detector == 1 % the one means, PLUG IN FRIDGE
        % this function estimates the rest of the power
        % this function estimates the electrical energy of the fridge
        % for 24h in kJ (eigentlich Leistung)
        obj.increased_demand_heat_power_day = ...
          obj.energy_consumption_day * obj.compressor_quotient * ...
          obj.factor_cooling_reducer * obj.epsilon * 3.6 / ...
          12 - 2 * 3.6 * obj.averaged_transmission_losses;
      else
        % COMBINE FRIDGE
        obj.increased_demand_heat_power_day = (obj.cooling_power * ...
           obj.factor_cooling_reducer - ...
           obj.averaged_transmission_losses) * 3.6; %
      end % if end

  end % function constructor end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% ordinary object specific functions %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  function r = sum_mass_times_capacity(obj)
  %% this function estimates a part of a main_equation
      % estimates mass specific heat-capacity
       r = sum(obj.specific_heat_capacity .* ...
    obj.mass_stored); % kJ/K
  end % function sum_mass_times_capacity end

  function [r, r_c_max]  = sum_transmission_losses(obj, ...
```

```
            number_steps, ...
            count_number_steps, ...
            count_number_day)
%% this function sums transmission losses of the wall
    % estimates sum of all wall transmission losses into fridge
    r = 0;
    r_c_max = 0;

    for i = 1:length(obj.modified_heat_transmission_coefficient)
     % it is necessary if temperature outside the fridges is constant
     % to get vector for each time step
     if length(obj.temperature_outside_fridge(i)) == 1
     n_temperature_outside_fridge = ...
     obj.temperature_outside_fridge(i) + zeros(1,number_steps);
     end % if end

     % VERY IMPORTANT estimates losses
     r = r + obj.modified_heat_transmission_coefficient(i) ...
     * (n_temperature_outside_fridge(count_number_steps) - ...
     obj.temperature_history(count_number_steps, count_number_day));
     % estimates maximal losses
     r_c_max = r_c_max + ...
     obj.modified_heat_transmission_coefficient(i) ...
     * (n_temperature_outside_fridge(count_number_steps) - ...
     obj.cooling_room_temperature_max);

    end % for end

    r = r * 3.6; % transmission losses in kJ (in actual fact, POWER)

end % function sum_transmission_losses end

function [r, time_to_T_critical_max]  = sum_all_losses(obj, ...
            number_steps, ...
            count_number_steps, ...
            count_number_day, ...
            number_days)
%% this function estimates the sum of the losses which can be
%  the losses can be positive or negative

    % sum all fridge losses for step
    [sum_transmission_losses, sum_max_transmission_losses] = ...
              obj.sum_transmission_losses( ...
              number_steps, ...
              count_number_steps, ...
              count_number_day);

    % here the time_no_cooling or time to the temperature go critical
    % will be estimated

    % logarithmic estimation of maximal transmission losses
    Q_T_log_max = (sum_max_transmission_losses - ...
         sum_transmission_losses) / ...
```

```
          log(sum_max_transmission_losses / ...
          sum_transmission_losses);

    % estimation of time till maximal temperature will be achieved
    time_to_T_critical_max = ((obj.cooling_room_temperature_max - ...
          obj.temperature_history(...
          count_number_steps, ...
          count_number_day)) * ...
          obj.sum_mass_times_capacity) / ...
          (Q_T_log_max * 3.6);

    % inspect what kind of day hour is it. Necessary because of
    % different level of losses
    if count_number_steps <  9 || count_number_steps >  20
        % here only the transmission losses influencing the
        % temperature
        r = sum_transmission_losses; %
    else
        % here in addition to the transmission losses the static
        % losses  of day activities influencing the temperature
        r = sum_transmission_losses + ...
            obj.increased_demand_heat_power_day;

        time_to_T_critical_max = ...
              ((obj.cooling_room_temperature_max - ...
              obj.temperature_history(count_number_steps, ...
              count_number_day)) * obj.sum_mass_times_capacity) / ...
              (Q_T_log_max * 3.6  + ...
              obj.increased_demand_heat_power_day);
    end % if else end
end % function sum_all_losses end

function r = capacity_estimator(obj, ...
          count_number_steps, ...
          count_number_day, ...
          Q_losses)
%% this function estimates the heat power, which can be
% stored into the fridge if the fridge temperature is T(i).

    r = Q_losses - (obj.cooling_room_temperature_min - ...
       obj.temperature_history(count_number_steps, ...
       count_number_day)) * obj.sum_mass_times_capacity; % in kJ

end % function capacity_estimator end

function r = temperature_change(obj, ...
          number_steps, ...
          count_number_steps, ...
          count_number_day, ...
          cooling, ...
          Q_losses)
%% this function estimates the temperature in the fridge
    Q = 0.8 * (Q_losses - cooling);
```

```
      %% this is the main equation
      r = Q / obj.sum_mass_times_capacity + ...
      obj.temperature_history(count_number_steps, ...
            count_number_day);
      %%
      if count_number_steps == number_steps
        obj.temperature_history(1,count_number_day + 1) = r;
        obj.electric_power_consumption_history(1, ...
            count_number_day + 1) = cooling / obj.epsilon;
      else
        obj.temperature_history(count_number_steps + 1, ...
           count_number_day) = r;
        obj.electric_power_consumption_history( ...
           count_number_steps + 1, count_number_day) =  ...
           cooling / obj.epsilon;
      end % if else end
  end % function main equation end
 end % methods end
end % class end
```

Literaturverzeichnis

[1] Angermann, Beuschel, Rau und Wohlfarth. *MATLAB - Simulink - Stateflow: Grundlagen, Toolboxen, Beispiele*. Oldenbourg, München, 6., aktualis. aufl. edition, 2009. 29, 34, 51

[2] Jaime Arias. *Energy Usage in Supermarkets - Modelling und Field Measurements*. text, Royal Institute of Technology, KTH, Energy Technology ; Stockholm, 2005. 3, 7, 51

[3] Bernhard Lahres und Gregor Rayman. *Objektorientierte Programmierung. das umfassende Handbuch*. Galileo computing. Galileo Press, Bonn, 2., aktualisierte und erw. aufl. edition, 2009. 4, 51

[4] EnergieAgentur.NRW. Energieeffizienz im Lebensmittel-Einzelhandel. URL: `http://www.energieagentur.nrw.de/unternehmen/page.asp?TopCatID=3695&CatID=3721&RubrikID=3743` (Zugriff am 20. Juli 2011). 3, 51

[5] Michael Kauffeld. *Stand der Technik von Supermarktkälteanlagen. Umwelteinfluss und Entwicklungspotential*. 2008. 9

[6] Martin Kleimaier. Netzintegration von Strom aus erneuerbaren Energiequellen: Zunehmende dezentrale Einspeisung erfordert eine Umrüstung der Netze. *Energy 2.0*, 1(7):45–47, 2008. 51

[7] F.A.T.M. Ligthart. Untersuchung zur Möglichkeit einer Abdeckung von Kühl- und Tiefkühlmobiliar in Supermärkten . Technical Report ECN-E–08-009, ECN Energy in the Built Environment, 2008. 3

[8] Elke Lorenz, Johannes Hurka, and Detlev Heinemann. Solarleistungsvorhersage zur netzintegration von solarstrom. *24. Symposium Photovoltaische Solarenergie : 04. - 06. März 2009, Kloster Banz, Bad Staffelstein*, page 6, 2009. 2, 51

[9] Sean McGowan. Supermarket refrigeration going natural. *HVAC&R Nation*, pages 8–9, 2007. 3, 51

[10] Caroline Möller. Spezifikation und Simulation einer Kältelast mit Kältespeicher im Energieversorgungsnetz. Diplomarbeit, Technische Universität Berlin, Berlin, Juli 2010. 7, 33, 34, 51

[11] P. Pepper. *Programmieren lernen: Eine grundlegende Einführung mit Java*. Springer, 2008. 4

[12] Arnd Poetzsch-Heffter. *Konzepte Objektorientierter Programmierung: Mit Einer Einführung in Java*. Springer, Berlin, 2009. 4, 51

[13] J Probst. Ein Netzwerk für effiziente Kältetechnik. URL: http://www.energieagentur.nrw.de/unternehmen/page.asp?TopCatID=3695&CatID=3721&RubrikID=3743 (Zugriff am 20. Juli 2011), 2009. 3

[14] Risto Ciconkov und Arnd Hilligweg. Simulationsprogramme für Kälteanlagen Einsatz in Ausbildung und Praxis. *Technik im Bau, Fachzeitschrift für Technische Gebäudeausrüstung*, 77(3):64–70, 2004. 7, 34, 51

[15] W. Schweizer. *MATLAB kompakt*. Oldenbourg Wissensch.Vlg, 2009. 4

[16] F. Steimle, H. Kruse, E. Wobst, and et al. Energiebedarf für die technische Erzeugung von Kälte. Technical Report Statusbericht Nr. 22, Deutscher Kälte- und Klimatechnische Verein (DKV) e.V., Stuttgart, 2002. 3

[17] Steve Völler. *Optimierte Betriebsführung von Windenergieanlagen durch Energiespeicher Elektronische Ressource*. Universitätsbibliothek Wuppertal, Wuppertal, 2010. 2, 51

[18] Michael Weigend. *Objektorientierte Programmierung mit Python 3. Einstieg, Praxis, professionelle Anwendung*. mitp-Verl., Heidelberg u.a., 4., aktualisierte aufl. edition, 2010. 4, 51