Projektgruppe 432

DEZENTEN

Ein erstes dezentrales
Energiemanagment-System

Endbericht

Sommersemester 2003
Wintersemester 2003/04

Teilnehmer:
Frederic A. Folwaczny, Markus Heinz, Dragan Isakovic, Martin Krogmann,
Stefan Nobis, Ralf Paaschen, Martin Piayda, Mykhaylo Rudermann, Andreas Schröder,
Carsten Sommer, Gerd Terhardt, Andreas Volgmann

Betreuer:
Horst F. Wedde, Frank Thorsten Breuer, Wolfgang Freund

Fachbereich Informatik
Universität Dortmund

Februar 2004

Inhaltsverzeichnis

Teil I.

Einleitung

1. Vorwort der Betreuer

Vor dem Hintergrund der Entflechtung der elektrischen Energieversorgung, des Erstarkens dezentraler Energieerzeugung, ökologischer und ökonomischer Aspekte haben sich der Inhaber des Lehrstuhls für Energiesysteme und Energiewirtschaft und der Inhaber des Lehrstuhls für Betriebssysteme und Rechnerarchitektur und ihre Mitarbeiter zu einem gemeinsamen Projekt (DEZENT) zusammengefunden. Das Ziel dieses Projektes ist der Nachweis, mittels eines kooperierenden Multiagentensystems, das europäische Energieverorgungsnetz dezentral führen zu können, und zu untersuchen, inwieweit ökologische und ökonomische Zielfunktionen im Energiemanagement durch eine verteilte dezentrale Energieumwandlung in Einklang zu bringen sind. Da dieses Ziel für alle Beteiligten bedeutete, Neuland zu betreten, entschieden sich die Veranstalter, eine Projektgruppe zu diesem Thema anzubieten. Diese Projektgruppe war also sofort als Ausgangspunkt für anschießende Projektaktivitäten gedacht.

Zum Inhalt der Projektgruppe seien zunächst die wichtigsten Punkte aus dem PG-Antrag genannt: knapper werdende fossile Energieträger, zunehmende Nutzung dezentraler regenerativer Energiequellen (Brennstoffzellen, Photopholtaik, Windkraft), dezentrale Steuerung von Energieerzeugung und -verteilung zur Erfüllung lokaler Bedürfnisse und Möglichkeiten, flexible Preisgestaltung, Konzeption und Implementation eines Systems verteilter Agenten, unvorhersehbare Verfügbarkeit von überschüssiger elektrischer Energie, unvorhersehbarer Bedarf an elektrischer Energie, finanzielle Kapazität der Konsumenten, Anforderungen der Konsumenten an Qualität, Komfort und Zuverlässigkeit im Sinne von Mindestquanten in einem Zeitrahmen (Lastkurven), Möglichkeiten der Produzenten bezüglich physikalischer Grenzen (zeitlicher Verfügbarkeit, Kennlinien), Adaptivität der Steuerung bei Ausfall von elektrischen Betriebsmitteln, bei unvorhersehbarem Bedarf oder Überschuss, komplett integrierte und adaptive Betriebssystem-Funktionen, Transaktionsmanagement.

Neben den inhaltlichen Zielsetzungen lag den Veranstaltern besonders

am Herzen, durch die Teilnahme an der Projektgruppe zwölf Experten für dieses moderne, immer wichtiger werdende interdisziplinäre Thema auszubilden, die (abgesehen von einer möglichen späteren beruflichen Tätigkeit ausserhalb der Universität) durch Diplomarbeiten und Mitarbeit das in Zusammenarbeit mit dem Lehrstuhl für Energiesysteme und Energiewirtschaft inizierte Forschungsprojekt unterstützen.

Die inhaltliche und auch die organisatorische Arbeit wurde von den Teilnehmern engagiert und größtenteils eigenverantwortlich abgewickelt. Die Veranstalter sind überzeugt, dass die gesteckten Ziele der Projektgruppe gut erreicht wurden. Die investierte Zeit hat sich offensichtlich für alle Beteiligten gelohnt. Auch die Hoffnung der Veranstalter, dass die Projektgruppe zu einem Keim für weitere Projektaktivitäten (DFG-Antrag) des Lehrstuhls wird, scheint sich zu bestätigen.

2. Projektgruppe 432

Projektgruppen sind ein fester Bestandteil des Informatikstudiums an der Universität Dortmund. Diese Lehrveranstaltung des Hauptstudiums soll vor allem praktische Kenntnisse (Hard- und Soft-Skills) vermitteln und eine Ergänzung zum eher theoretisch orientierten Lehrangebot an einer Universität darstellen.

Eine Projektgruppe besteht aus zehn bis zwölf Studentinnen und Studenten[1] und in der Regel zwei Betreuern. Die Studenten sollen in dieser Gruppe in einem Zeitraum von zwei Semestern selbständig ein vorgegebenes, umfangreiches Thema bearbeiten. Dokumentiert wird diese Arbeit in zwei Dokumenten, dem Zwischenbericht zum Abschluss des ersten Semesters und dem Endbericht nach Abschluss des zweiten Semesters. *Das vorliegende Dokument ist der Endbericht der Projektgruppe 432* Dezenten.

2.1. Thema der Projektgruppe

Die Projektgruppe Dezenten verfolgt den Ansatz ein *erstes dezentrales Energiemanagement-System* durch ein *Multiagentensystem* zu realisieren.

Vor dem Hintergrund knapper werdender fossiler Energieträger und zunehmender Klimaveränderungen wird in Deutschland die Nutzung regenerativer Energiequellen (Windkraftanlagen und Solarzellen) stark subventioniert. Die seit Jahren zunehmende Anzahl dieser Anlagen macht den Netz- und Kraftwerksbetreibern nur deshalb keine Probleme, weil ihre Gesamtleistung im Gegensatz zu der der Größtkraftwerke noch verschwindend gering ist. Nach allgemeiner Einschätzung wird sich der Anteil dezentraler

[1] Im folgenden wird wegen der einfacheren Lesbarkeit ausschließlich die männliche Form verwendet. Gemeint sind immer männliche und weibliche Form, außer es wird ausdrücklich darauf hingewiesen, dass nur die weibliche oder männliche Form gemeint ist.

Stromerzeugungsanlagen in den nächsten zehn Jahren verdoppeln. In etwa drei bis fünf Jahren sollen Brennstoffzellenkraftwerke in größeren Stückzahlen auf den Markt kommen. Damit wird die traditionell „top-down"-orientierte Stromversorgung auf den Kopf gestellt; potentiell sind nämlich dann alle Energieverbraucher oder Verbrauchergruppen auch Energieerzeuger.

Langfristig sind daher neue Informations- und Steuerungstechniken nötig, um mit vielen dezentralen Anlagen die Stromversorgung zu sichern. Gerade großen Energieversorgern macht ein scheinbarer Widerspruch zwischen dezentralen Systemen und dem Stand der Technik entsprechende Sicherheitsmaßstäbe in Betrieb und Wartung [VDE97] Kopfzerbrechen. So planen „die Energieversorger . . . , ihren Kunden Haus-Brennstoffzellenkraftwerke in den Keller zu stellen . . . und zentral zu steuern." [Har02] Das ist aber eine höchst unzuverlässige und finanziell aufwendige Kontroll-Lösung bei der die Energieversorger die Verantwortung zur störungsfreien Energielieferung gar nicht übernehmen könnten. Diese wäre ja gerade an dem Ort der Energieproduktion am besten zu gewährleisten. Es ist unschwer abzusehen, daß eine solche Strategie die Verbraucher wesentlich teurer zu stehen käme, während sie gleichzeitig keine Kontrolle über die Preise hätten.

Die Projektgruppe hat nun als Ziel, für eine dezentrale Steuerung von Energieerzeugung und -verteilung die den lokalen Bedürfnissen und Möglichkeiten angepaßte flexible Preisgestaltung einzurichten.

2.2. Mitglieder der Projektgruppe

Die Projektgruppe DEZENTEN wird veranstaltet und betreut von

- Prof. Dr. Horst F. Wedde
- Dipl.-Inform. Frank Thorsten Breuer
- Dipl.-Ing. Wolfgang Freund.

Die Mitglieder der Projektgruppe sind (in alphabetischer Reihenfolge):

- Folwaczny, Frederic A.

- Heinz, Markus
- Isakovic, Dragan
- Krogmann, Martin
- Nobis, Stefan
- Paaschen, Ralf
- Piayda, Martin
- Rudermann, Mykhaylo
- Schröder, Andreas
- Sommer, Carsten
- Terhard, Gerd
- Volgmann, Andreas

2.3. Ziele der Projektgruppe

Ziel der Projektgruppe ist die Konzeption und Implementation eines Systems verteilter Agenten, die durch Kommunikation (Verhandlungen, Transaktionen) Anforderungen und Wünsche ihrer Klienten nach elektrischer zentral und dezentral erzeugter Energie und Wärme (Solar- bzw. Brennstoffzellen [VDI02] und Kraft-Wärme-Kopplungen) erfüllen. Im Rahmen der Möglichkeiten einer Projektgruppe beschränkt sich die Arbeit auf die Preisgestaltung. Hierbei sollen folgende Randbedingungen beachtet werden:

- unvorhersehbare Verfügbarkeit von überschüssiger elektrischer Energie
- unvorhersehbarer Bedarf an elektrischer Energie
- finanzielle Kapazität der Konsumenten

- Anforderungen der Konsumenten an Qualität, Komfort und Zuverlässigkeit im Sinne von Mindestquanten in einem Zeitrahmen (Lastkurven)
- Möglichkeiten der Erzeuger bezüglich physikalischer Grenzen: Kennlinien, zeitliche Verfügbarkeit
- Adaptivität der Steuerung bei Ausfall von elektrischen Betriebsmitteln

Betriebsmittel sollen als Ressourcen und Agenten in einem verteilten System realisiert werden, z. B.:

- Haushalte, Industriebetriebe als Abnehmer von Energie
- Solaranlagen, Brennstoffzellenkraftwerke und Windkraftanlagen als dezentrale Energieerzeuger
- verschiedene Kraftwerke als zentral-gesteuerte Energieerzeuger nur zum Ausgleich lokaler/regionaler Engpässe
- Leitungssegmente, Umspannstationen, Trennstellen als Transportwege

Ausschlaggebende Komponenten sind dem Bedarf entsprechende Betriebssystem-Funktionen und zuverlässige Realzeit-Kommunikationsmethoden.

2.3.1. Minimalziel

Im Rahmen der Projektgruppe sollen mindestens folgende Funktionseigenschaften und Komponenten realisiert werden:

- zuverlässige Kommunikation zwischen Agentenprozessen
- minimale Rollenaufgaben der folgenden Agenten
  - Haushalte, Industriebetriebe als Abnehmer von Energie
  - Solaranlagen, Brennstoffzellenkraftwerke als dezentrale Energieerzeuger
  - Netzbetreiber als Lieferant oder Abnehmer von Energie

2.4. Entwicklung von Dezenten in zwei Iterationsschritten

Bis vor wenigen Jahren war es üblich, das erste Semester einer Projektgruppe zur Wissensaquisition zu nutzen. Diese umfasste nicht nur – wie auch in dieser Projektgruppe – eine Seminarphase, sondern auch einige oder mehrere Praktikumsphasen. Diese dienten der vertiefenden und praktischen Einarbeitung in das Themengebiet und stellten sicher, dass sich alle Teilnehmer in Theorie und Praxis auf einem vergleichbaren Kenntnisstand befinden. Das in den Praktikumsphasen erworbene Wissen sollte es ermöglichen, die Phasen der Softwareentwicklung nur einmal durchlaufen zu müssen. Probleme, die erst nach der Designphase erkennbar wurden, waren so nur schwer zu lösen.

Auf Anraten des Softwaretechnologielabors (STL) des Fachbereichs verfolgt die Projektgruppe einen inkrementellen Ansatz, der bei der Entwicklung von MELODY erfolgreich eingesetzt wurde [WLE95] und auch Grundlage des vom STL empfohlenen Extreme-Programmings ist: Die Entwicklungsphasen werden mehrfach durchlaufen. Die Erkenntnisse der vorhergehenden Versionen gehen in die Entwicklung der folgenden ein.

Die erste Version wurde im Verlauf des ersten Semesters entwickelt. In dieser Phase der PG erfolgten die Einarbeitung im Rahmen der Seminarphase. Anschließend wurde eine Anforderungsanalyse durchgeführt, aus der ein Pflichtenheft für die erste Implementation erstellt wurde. Im Anschluss an die Implementation erfolgten die ersten Tests.

Die zweite Version wurde im Verlauf des zweiten Semesters erstellt. Hier sollten einige Verbesserungen vorgenommen werden, wie etwa die Vereinfachung der Bedienung über eine graphische Benutzeroberfläche. Außerdem sollten in der zweiten Version die Tests in größerem Umfang erfolgen, so dass mehr Erkenntnisse über das autonome Arbeiten der Agenten im System gewonnen werden können.

2.5. Zeitplan

2.5.1. Erstes Semester

28.02.2003	Erstes PG-Treffen
03.03.2003	Verteilung der Seminarthemen
24.04. - 25.04.2003	Seminarphase
07.05.2003	Firmenvorträge zum Thema Projektmanagement
08.05. - 09.05.2003	Seminar des Softwaretechnologie-Labors zur Einführung in Entwicklungstools
12.05. - 16.05.2003	Anforderungsspezifikation
19.05. - 20.05.2003	Designphase
23.05. - 25.06.2003	Implementierung
28.05. - 22.07.2003	Integrations- und Testphase
16.07.2003	PG-Grillparty
18.07. - 29.07.2003	Fertigstellung Zwischenbericht

2.5.2. Zweites Semester

13.10. - 17.10.2003	Durchführung von Experimenten mit der erstellten ersten Version
20.10. - 31.10.2003	Anforderungsspezifikation
03.11. - 15.11.2003	Designspezifikation
17.11. - 12.12.2003	Implementierung
15.12. - 19.12.2003	Komponententest, Performanceanalyse
05.01. - 16.01.2004	Integration und Gesamttest
19.01. - 30.01.2004	Anwendung mit verschiedenen Lastprofilen
02.02. - 31.03.2004	Fertigstellung Endbericht
30.04.2004	Abschlusspräsentation

3. Seminarphase

Zur Einarbeitung in die für die Projektgruppe wichtigen Themengebiete wurde ein Seminar veranstaltet. Es fand am 24. und 25.04.2003 im Haus Ortlohn in Iserlohn statt.

Am Vormittag des ersten Tages wurden von Andreas Volgmann, Andreas Schröder und Dragan Isakovic Vorträge zu elektrotechnischen Gesichtspunkten der Projektgruppe gehalten. Am Nachmittag führten Gerd Terhard, Frederic A. Folwaczny und Martin Piayda in Agentensysteme ein, Ralf Paaschen hielt einen Vortrag über Softwareentwurf und -testmethoden. Am zweiten und letzten Tag stellten vormittags Carsten Sommer, Stefan Nobis und Mykhaylo Rudermann das MELODY-System vor. Am Nachmittag führten Markus Heinz und Martin Krogmann in Real-Time Linux ein.

Das *Haus Ortlohn* ist für ein Seminar zum Thema dezentrales Energiemanagement besonders interessant, weil der Wärmebedarf und ein Teil des Bedarfs an elektrischer Energie des Gebäudekomplexes durch ein Blockheizkraftwerk (BHKW) mit Kraft-Wärme-Kopplung, das auch zu besichtigen ist, gedeckt wird. Auch im Allgemeinen bot die Tagungsstätte der Veranstaltung den gebührenden Rahmen. Das in einer schönen, zu Spaziergängen einladenden Parklandschaft gelegene Seminargebäude bietet gut ausgestattete Konferenzräume verschiedener Größe. Das Essen kann als gut bürgerlich aber nicht schwer bezeichnet werden und bietet auch den Anhängern vegetarischer Kost die Möglichkeit, ihr Gewicht zu halten. Die Übernachtungsmöglichkeiten sind schlicht aber zweckmäßig eingerichtet und wurden als angenehm empfunden, ebenso die Örtlichkeiten für ein fröhliches Beisammensein und Kennenlernen.

Die Tagungsstätte kann – nachdem sich Seminarfahrten zu Tagungsstätten der ev. Kirche in Westfalen für Projektgruppen des Fachbereichs beinahe zu einer Tradition entwickelt haben – auch nachfolgenden Projektgruppen weiterempfohlen werden. Die Adresse lautet:

Ev. Kirche von Westfalen
Haus Ortlohn
Berliner Platz 12
58638 Iserlohn
Telefon: (0 23 71) 3 52-0
Telefax: (0 23 71) 3 52-299

In den folgenden Abschnitten befinden sich die Zusammenfassungen der einzelnen Vorträge. Die vollständigen Ausarbeitungen können im Seminarband (siehe [FHI+03a]) nachgelesen werden.

3.1. EI. Versorgungsnetze und deren Management

Andreas Volgmann

Dieser Vortrag beschäftigt sich mit der Energieversorgungsstruktur der heutigen Zeit. Die Anforderungen an die Qualität der Energie wachsen immer mehr. Gleichbedeutend damit sind möglichst geringe Verluste der Energie und deren Zuverlässigkeit. Diese Anforderungen gehen teilweise so hoch, dass man sich einen Stromausfall nicht mehr leisten kann, wie z.B. in Krankenhäusern oder auch bei großen Computeranlagen. Der Aufbau unserer elektrischen Versorgungsnetze sowie die Netzstruktur werden hier näher beschrieben. Darüber hinaus wird auch auf kurzfristige Strombeschaffungsmöglichkeiten eingegangen, sowie auf Lastspitzen.

Um eine möglichst hohe Ausfallsicherheit zu gewährleisten müssen elektrische Versorgungsnetze dementsprechend geplant werden. Dazu gehört auch die richtige Dimensionierung des Netzes. Engpässe sollten möglichst nicht entstehen, darüber hinaus will man in großen Firmen den benötigten Strombedarf vorher planen können. Am Beispiel des Strommanagements von Krankenhäusern, sowie am Management der Bayer AG wird dies an zwei konkreten Beispielen verdeutlicht.

3.2. Übersicht über die Möglichkeiten dezentraler Stromerzeugung

Andreas Schröder

Seit einigen Jahren schon entwickelt sich die dezentrale Stromerzeugung – meist aus regenerativen Energien – von einer Randerscheinung zu einem immer wichtigeren Bestandteil der Energieversorgung. Zu den hier betrachteten Verfahren zählen neben der schon seit längerer Zeit angewandten Wind- und Wasserkraft auch die Erzeugung von Strom mit Hilfe der Photovoltaik, Brennstoffzellen und Biomasseanlagen.

Bisher werden solche Anlagen jedoch nur als Randerscheinung betrachtet und nicht vollständig in das Gesamtkonzept der elektrischen Energieversorgung integriert. Stromversorgungsunternehmen sehen die Möglichkeiten dezentraler Erzeuger im Bereich der Spitzenlastabdeckung und der Verbesserung der Netzqualität bzw. schaffen Probleme, da sie sich nicht zentraler steuern lassen.

Ein Ansatz der Industrie ist es, durch zentrale Steuerstellen eine größere Menge von Erzeugern in Form von virtuellen Kraftwerken zu managen. Ansätze, die den Besitzern kleiner Anlagen erlauben am Strommarkt zu partizipieren, sind bisher nicht zu erkennen. Die heutige Attraktivität der dezentralen Stromerzeugung ist zu großen Teilen nicht durch eine effiziente Einbindung, sondern durch die Förderung der neuen Technologien mit staatlichen Hilfen, insbesondere den garantierten Einspeisevergütungen, zu erklären.

Damit eine bessere Integration möglich ist, muss besondere Rücksicht auf die Eigenheiten der Energieerzeugung mit Hilfe der verschiedenen dezentral verwendeten Technologien, wie Leistungsfähigkeit, Verfügbarkeit, Reaktionszeit, etc. Rücksicht genommen werden.

3.3. Kostenmanagement el. Energieversogung

Dragan Isakovic

Um das Kostenmanagement der Energieerzeugung zu verstehen, muss man sich vor Augen führen, dass Energieerzeugung durch die Beschaffung der

Anlagen, deren Betrieb und Wartung Kosten entstehen. Die Disziplin der Betriebswirtschaftslehre ist es, diese Kosten zu erkennen, zu strukturieren und Methoden zu entwickeln, Kosten transparent und verständlich zu kennzeichnen.

In der Ausarbeitung wird darauf eingegangen, welche Kostenarten bei der Energieerzeugung entstehen, ebenso welche Kostenstellen es in diesem Fall gibt. Prinzipien der Investitionsrechnung sind auch erklärt. Da die Kosten letztendlich auf die Verbraucher abgetragen werden, die das Gut Energie beziehen, werden Verbraucherkosten und die verschiedenen Ausprägungen der Tarife erläutert.

Da Energie eine der bedeutendsten Handelsgüter überhaupt darstellt – nicht allein durch das Volumen welches produziert und vertrieben wird – hat man sich entschlossen, die erzeugte Enrgie auf Energiebörsen zu handeln. Deshalb wird das Prinzip der größten und wichtigsten Enegiebörse der BRD beschrieben.

Im Hinblick auf die Relevanz für die Projektgruppe, gibt die Ausarbeitung eine Sicht auf den momentanen Trend beim Kostenmanagement dezentraler Energieerzeuger. Es werden Vorteile solcher Anlagen bei der Kostensenkung und Umweltverträglichkeit aber auch Nachteile bei deren Einführung auf dem Markt erklärt.

3.4. Einführung in Multi-Agenten-Systeme

Gerd Terhardt

Diese Einführung in Multi-Agenten-Systeme soll einen ersten Überblick über die Vielfalt unterschiedlichster Agentenarchitekturen geben.

Da es kaum einheitliche Definitionen in diesem Bereich gibt, wird zunächst eine gemeinsame Basis über die zentralen Begriffe Agent und Multi-Agenten-Systeme geschaffen. Darauf aufbauend werden konkrete Architekturen mit ihren unterschiedlichen Schwerpunkten und Anwendungsgebieten vorgestellt. Die zur Darstellung der Multi-Agenten-Systeme benötigten Sprachen werden anschliessend umrissen.

Ein wichtiger Bereich bei Multi-Agenten-Systemen behandelt die Frage der Interaktion und Kooperation der einzelnen Agenten miteinander. Im Bereich der Interaktion werden die Voraussetzungen und Kriterien erläu-

tert. Ebenso werden Kooperationsformen vorgestellt, die sich an menschlichen Organisationen orientieren. Damit die Agenten gemeinsam agieren können, müssen sie zusätzlich ihr Verhalten aufeinander abstimmen. Dies geschieht über Planungsmechanismen, wobei das zentrale, dezentrale und Blackboard-Planen näher betrachtet werden.

Um nun auch dynamische Aktionen darstellen zu können, benötigt man Modellierungsmöglichkeiten, wobei auf die Prozessdefinition in der Informatik zurückgegriffen wird. Neben weiteren sind die zustands- und objektoriente Modellierung von zentraler Bedeutung.

In einem Agentensystem müssen die Agenten in der Lage sein, Informationen untereinander austauschen zu können. Die erforderliche Kommunikation kann mittels KQML, einem speziellen Kommunikationsprotokoll, für das eine Einführung gegeben wird, geschehen.

Abschliessend wird auf die Koordination der Aktionen eingegangen, die im Gegensatz zur Kooperation die individuellen Aufgaben der Agenten definiert, um ein effektives Gesamtsystem zu gewährleisten. Erst durch die Koordination wird eine Zusammenarbeit der Agenten möglich.

3.5. Preisverhandlungen in Multi-Agenten-Systemen

Frederic A. Folwaczny

Das Thema der Preisverhandlungen in Multi-Agenten-Systemen ist ein zentrales Thema für die Projektgruppe Dezenten. Im Rahmen dieser Arbeit wird das Thema in drei Schritten nähergebracht.

Zunächst wird die Preistheorie, ein weit erforschtes Thema der Volkswirtschaftslehre, kurz umrissen. Es wird zunächst aus dem Bereich der mikroökonomischen Theorie dargestellt, wie ein Preis für ein Gut zustande kommt. Anschließend wird der Verlauf von Angebot und Nachfrage als Funktion des Preises erörtert. Das Angebot steigt mit dem Preis, während die Nachfrage fällt. Der Preis, bei dem Angebot und Nachfrage übereinstimmen, ist das Marktgleichgewicht.

Danach werden die Verhandlungen und die Verfahren zur Aufgabenverteilung in Agentensystemen dargestellt. Als Anwendungsbeispiele für Aufgabenverteilungen werden Auktionsmechanismen dargestellt.

Abschließend wird ein Experiment mit einem Agentensystem in der Logistik beschrieben. Der Kernpunkt dieses Experimentes ist die Untersuchung der Wirkung einer Option, die das Zurücktreten aus abgeschlossenen Verträgen behandelt. Diese Möglichkeit erhöht maßgeblich den Gewinn der Firmen, die diese Option benutzen, sowie die Effizienz des Gesamtsystems.

3.6. Realzeitagenten

Martin Piayda

Realzeitagenten erhalten zunehmende Bedeutung, wenn es um die Frage nach Zuverlässigkeit, Verfügbarkeit und Genauigkeit von Ergebnissen geht. Heutige Umgebungen sind auf Ergebnisse angewiesen, die zu festgelegten Zeitpunkten zur Verfügung stehen und eine logische Korrektheit aufweisen müssen. Realzeitagenten erweitern die klassischen Agenten und Multiagenten um Techniken, die dieses leisten.

Es ist notwendig, den Begriff *Realzeit* zu definieren, und dabei Techniken zu betrachten, die Realzeit-Verhalten überhaupt erst möglich machen. Hierzu gehen wir im Vortrag auf wichtige Elemente, wie das Einhalten von Zeitschranken, Synchronisationsmodelle, der Nachrichtenaustausch und Prozessmodelle ein. Da sich Realzeitagenten noch im Forschungsstadium befinden, wird im Seminarvortrag der *Time-Triggered Message-Triggered Objects*-Ansatz (TMO) für Realzeitagenten betrachtet, und wie diese Agenten in ihre Umgebung eingebracht werden und miteinander kommunizieren.

3.7. Softwareentwurf und -testmethoden

Ralf Paaschen

Der Bereich der Projektintegration beschäftigt sich hauptsächlich mit der Planung und Durchführung eines Softwareprojektes. Hier werden das Wasserfallmodell und der „Rational Unified Process“ mit der Sprache UML vorgestellt.

Im Rahmen des Testens von Software wird zuerst der Begriff der „Qualität“ eingeführt. Nachfolgend werden verschiedene Ursachen für Fehler

untersucht und die Bedeutung eines sauberen Entwurfs für die Implementierung hervorgehoben. Danach werden statische und dynamische Prüfverfahren vorgestellt sowie die gezielte Erstellung von Testdaten. Hierzu gehört insbesondere auch die Methode der Äquivalenzklassenbildung. Hierbei werden auf Basis einer informellen Spezifikation systematisch Äquivalenzklassen gebildet und mögliche Entgabewerte erstellt, mit denen getestet werden kann.

3.8. Incremental Experimentation und Melody Entwicklungsphasen

Carsten Sommer

An das von der Projektgruppe zu entwickelnde Agentensystem werden hohe Anforderungen hinsichtlich Echtzeitfähigkeit und Sicherheit gestellt. Allerdings scheinen klassische Entwicklungsmethodiken für Echtzeitsysteme in diesem Fall ungeeignet. Das begründet sich zum einen in der verteilten Natur des Systems, zum anderen in der Unvorhersagbarkeit von äußeren Einflüssen, die auf das System einwirken. Klassische Echtzeitsysteme werden jedoch gerade für gut vorhersagbare Umgebungen entwickelt.

Aus diesem Grund wird hier eine neue Entwicklungsmethodik vorgestellt: Incremental Experimentation. Mit Hilfe dieser Methodik wurde MELODY, ein verteiltes Echtzeitbetriebsystem für echtzeitkritische Anwendungen entworfen. Anhand der verschiedenen Entwicklungsphasen, die MELODY durchlaufen hat, wird deutlich gemacht, wie Incremental Experimentation eingesetzt wurde, um ein System zu entwerfen, das adaptiv, auf die in Konflikt stehenden Anforderungen, reagiert.

Die bei der Entwicklung von MELODY gemachten Erfahrungen legen es nahe, sich auch bei der Entwicklung des Agentensystems für die Projektgruppe an Incremental Experimentation als Entwicklungsmethodik zu orientieren.

3.9. Module des Melody-Systems

Stefan Nobis

MELODY ist ein verteiltes Echtzeit-Betriebssystem, das besonders auf die sicherheitskritischen Bedürfnisse in unvorhersehbaren Umgebungen zugeschnitten wurde. Da derartige Systeme mit widersprüchlichen Anforderungen wie Zuverlässigkeit, Fehlertoleranz und Flexibilität zu kämpfen haben, sind MELODY-Systemdienste in großem Umfang adaptiv ausgelegt.

MELODY besteht aus den folgenden vier Kernkomponenten: dem *Task Scheduler*, der (nicht-präemptiv) Rechenzeit zuteilt, dem *Run-Time Monitor*, der die Arbeit des Task Schedulers unterstützt und Tasks gegebenenfalls abbricht, dem *File Server*, der den Zugriff auf verfügbare Ressourcen (Dateien) regelt, und dem *File Assigner*, der für die Verwaltung der verteilten Ressourcen zuständig ist.

Der neuartige Ansatz, der bei MELODY verfolgt wurde und den Run-Time Monitor notwendig macht, ist die Umkehr der Reihenfolge von Resourcen- und Taskplanung. So wird, anders als in herkömmlichen Ansätzen, zuerst das Task-Scheduling durchgeführt und erst anschließend versucht der Task, seine benötigten Ressourcen zu reservieren.

3.10. Melody-Transaktionsmanager

Mykhaylo Ruderman

In einem sicherheitskritischen Echtzeitsystem bestehen die Transaktionen aus den einzelnen Tasks. Damit eine Transaktion erfolgreich wird, müssen alle ihre Tasks erfolgreich ausgeführt werden, dabei soll die Transaktion ihre Deadline nicht verpassen. Ein adaptives Transaktionsmanagement in MELODY sorgt für die optimale Tasksausführung gemäß ihren *criticality*, *sensitivity* und *similarity*. *Similarity* ist ein Mass dafür, wie lang die Datenobjekte als "genau genug" betrachtet werden können. So werden unter der Berücksichtigung von *similarity* nur die benötigten Tasks der Transaktion ausgeführt, wodurch Systemresourcen gespart und die Systemstabilität erhöht werden.

Nach der Einführung in das Transaktionskonzept für die sicherheitskritischen Anwendungen, wobei die Begriffe *criticality*, *sensitivity*, *similarity*

und *deadline* erläutert werden, werden Aufbau und Ausführungsverlauf der Transaktionen beschrieben. Anschließend werden das Systemmodell und die Protokolle für *adaptive concurrency control* präsentiert.

3.11. Real-Time Linux

Markus Heinz, Martin Krogmann

Dieser Seminarbeitrag behandelt einige Themen aus dem Bereich der Echtzeitbetriebssysteme. Hierzu wird zuerst geklärt, was ein Echtzeitbetriebssystem ist, was die technischen Schwierigkeiten dabei sind und welche Lösungsansätze es für diese Probleme gibt. Zwei auf Linux basierende Lösungen, RT-Linux und RTAI, werden dann detailierter beschrieben. Es wird eine Übersicht über die allgemeine Architektur der beiden Systeme, deren API, deren Scheduler und deren Möglichkeiten der Interprozesskommunikation gegeben. Bei der Interprozesskommunikation wird das LXRT Modul von RTAI detaillierter beschrieben, da dies die Möglichkeit bietet, harte Echtzeitprozesse im Benutzeradressraum auszuführen, was in anderen Systemen meist nicht möglich ist.

Als mögliche Anwendung für ein Echtzeitbetriebssystem wird die Uhrensynchronisation in einem verteilten System vorgestellt. Im Rahmen dieses Beitrages werden dann auch noch einige Synchronisationsprotokolle vorgestellt, wobei der Schwerpunkt auf dem Real-Time-Burst-Protocol liegt, welches ein Broadcast-Protokoll ist und sich in Tests als am Besten erwiesen hat.

Teil II.

Konzeption und Implementation – Erste Phase

Im folgenden wird die erste von zwei Entwicklungsphase von DEZENTEN kurz beschrieben. Die Vorgehensweise wurde bereits in 2.4 erläutert.

Der Schwerpunkt soll hierbei auf der Anforderungsanalyse und den Tests liegen, die Implementation wird nur kurz behandelt. Für eine detailliertere Beschreibung der ersten Version von DEZENTEN ist der Zwischenbericht [FHI+03b] hinzuzuziehen.

4. Anforderungshandbuch

Die Simulation eines dezentralen Energiemanagement-Systems soll als Multiagenten-System realisiert werden. Die im folgenden aufgeführten Akteure sind durch Agenten zu repräsentieren.

- Bilanzkreisverantwortlicher (BKV)
- Verbundnetzbetreiber (VN)
- Blackboard (BB) als Energiebörse
- dezentrale Erzeuger elektrischer Energie
- Verbraucher elektrischer Energie

Die Struktur des Multiagenten-Systems ist in Abbildung 4.1 auf der nächsten Seite wiedergegeben. Es sind folgende Agenten zu implementieren:

- (kombinierter) Verbundnetz- und Bilanzkreis-Agent
- Blackboard-Agent
- Erzeuger-Agenten (EA) mit unterschiedlichen Technik-Modulen, die jede Art dezentraler Energieerzeugung simulieren
- Verbraucher-Agenten (VA) mit Simulationen von Verbrauchern mit unterschiedlichen Lastprofilen und unterschiedlicher Flexibilität

4.1. Erzeuger-Agent und Technik

Die Erzeuger-Agenten repräsentieren die Interessen der Betreiber dezentraler Energieerzeugungsanlagen. Die möglichen Anlagen sind:

- Brennstoffzellen

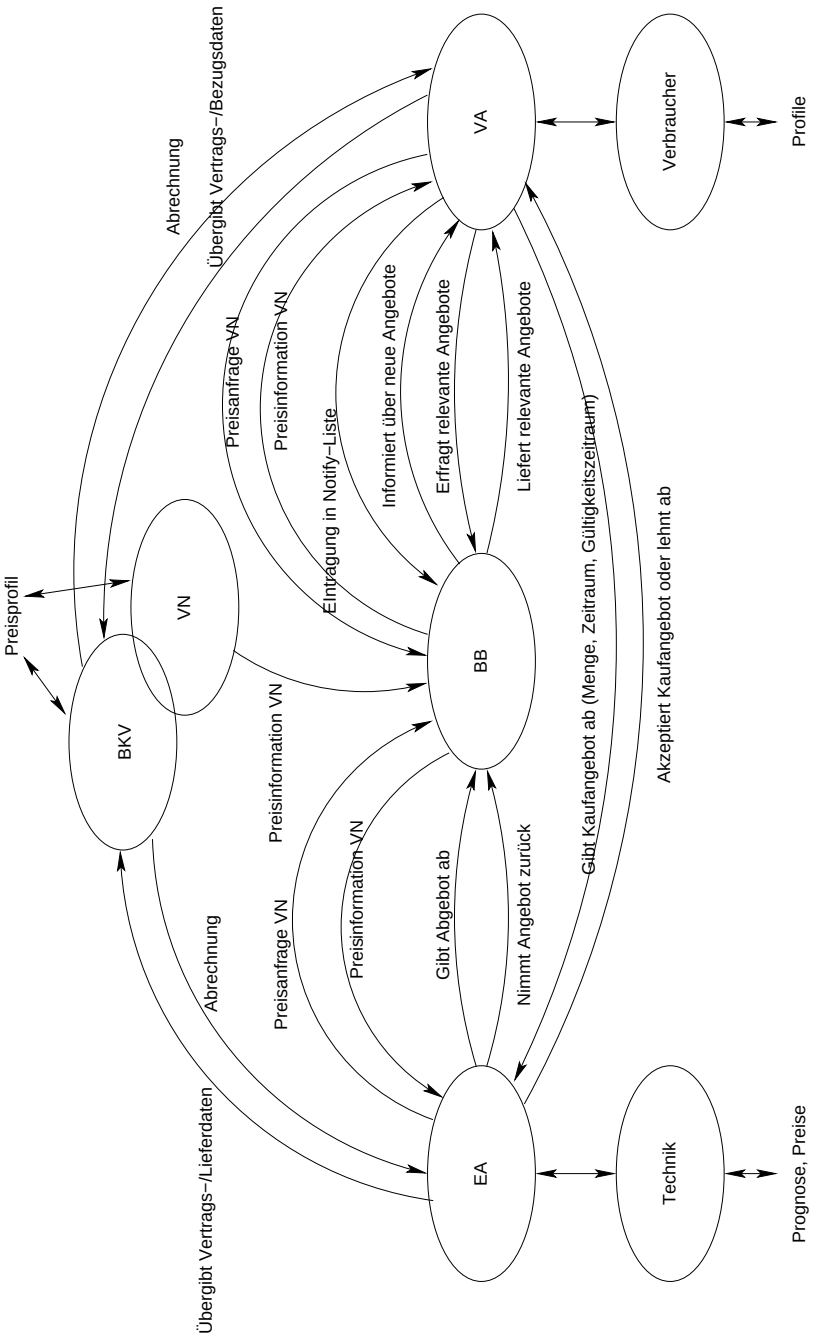

Abbildung 4.1.: Struktur des Agentensystems

- Solaranlagen
- Windkraftanlagen

Ziel der Erzeuger-Agenten ist es, Angebote über die Lieferung elektrischer Energie zu erstellen und sie Verbrauchern elektrischer Energie über ein Blackboard zugänglich zu machen. Diese Angebote können von den Verbrauchern zeitlich oder leistungs-gesplittet angenommen werden (zum Ablauf der Verhandlung siehe Abschnitt 4.4).

Brennstoffzellen: Liegt eine Brennstoffzelle vor, so kann diese prinzipiell jederzeit Leistung erzeugen. Für den EA ist nun entscheidend, ob er die Leistung zu einem höheren Preis verkaufen kann, als Kosten zur Erzeugung (Brennstoffkosten, Wartung, ...) entstehen. Um also einen vernünftigen Verkaufspreis $p_e(t)$ bestimmen zu können, benötigt er vom Technik-Modul den zeitlichen Verlauf der Kosten, die entstehen, um Strom zu produzieren. Dies entspricht einem Diagramm, in dem die Leistung je Zeitheit (z. B. $^{W}/_{s}$) gegen die Kosten aufgetragen wird. Dieses Diagramm ist in unserem Modell zeitunabhängig, da die Brennstoffkosten als konstant angenommen werden. Sollte der Erzeuger-Agent mit diesen Daten einen Vertrag abschliessen können, so gibt er die entsprechenden Daten (Zeitraum, Leistung) des Vertrages an das Technik-Modul weiter.

Wind-, Solarenergie: Da hier die Leistung nicht permanent abgerufen werden kann, benötigt der Erzeuger-Agent andere Informationen von der Technik. Zur Verhandlung braucht der Erzeuger-Agent den Zeitraum, in dem Leistung verkauft werden kann, die Leistung, die produziert wird, und die Qualität. Die Qualität entspricht einer Wahrscheinlichkeit, mit der die Leistung geliefert werden kann. Das Technik-Modul muss also die Wahrscheinlichkeiten für unterschiedliche Leistungen übermitteln. So könnte es z. B. sein, dass mit einer Wahrscheinlichkeit von 20% 50 Kilowatt, mit einer Wahrscheinlichkeit von 50% 30 Kilowatt und mit einer Wahrscheinlichkeit von 100% eine Leistung von 10 Kilowatt für denselben Zeitraum geliefert werden könnte. Aus diesen Daten kann der Erzeuger-Agent daraufhin ein Angebot über eine vernünftige Menge und zu einem vernünftigen Preis machen. Wiederum übergibt der Erzeuger-Agent die notwendigen Vertagsdaten an die Technik. Desweiteren sind dem Erzeuger-Agenten Kosten

für Wartung u. ä. zu übermitteln, die den Preis noch beeinflussen könnten.

Zur Abrechnung muß die Technik die tatsächlich eingespeiste Leistung zum Vertragszeitpunkt an den Erzeuger-Agenten leiten, damit die Abrechnung mit Verbraucher-Agent und Bilanzkreisverantwortlichem erfolgen kann, siehe Abschnitt 4.5.

4.2. Verbraucher-Agent und Verbraucher

Die möglichen Verbraucher werden durch Module beschrieben, wobei zwischen flexiblen und unflexiblen Verbrauchern (s. u.) unterschieden wird. Auf diese Weise können sowohl verschiedene Arten von Industrieunternehmen als auch die Gesamtheit aller Haushalte eines Bilanzkreises leicht modelliert werden. Einzelhaushalte spielen für diese Simulation keine Rolle, da deren Schwankungen im Energieverbrauch in der Masse untergehen, so dass alle Haushalte zu einem unflexiblen Verbraucher mit gut vorhersehbarem Lastprofil zusammengefasst werden können.

Die Verbraucher-Agenten erfahren von ihrem Verbraucher-Modul den prognostizierten Energiebedarf in Form eines Lastprofils (siehe Abschnitt 4.2.3), so dass bekannt ist, wann voraussichtlich wieviel Leistung benötigt wird. Anschließend versucht der Verbraucher-Agent den Energiebedarf des simulierten Verbrauchers durch Vertragsabschlüsse mit einem oder mehreren Erzeuger-Agenten zu decken. Um diese Erzeuger-Agenten und deren Angebote leichter zu finden, bedient sich der Verbraucher-Agent des Blackboards, von dem alle derzeit verfügbaren Angebote erfragt werden können.

Sollte auf diese Weise nicht der gesamte Energiebedarf des Verbrauchers durch dezentral organisierte Erzeuger gedeckt werden können, so besorgt der Verbraucher-Agent die fehlende Menge aus dem Verbundnetz, das zur Energielieferung jederzeit und ohne explizite Vertragsabschlüsse zu Verfügung steht.

Nach Ablauf des Vertragszeitraums findet die *Abrechnung* statt. Der Verbraucher-Agent rechnet die vertragsgemäß gelieferte Energie mit dem entsprechenden Erzeuger-Agenten ab. Abweichungen von den vereinbarten Leistungen werden mit dem Bilanzkreis abgerechnet, siehe Abschnitt 4.5.

4.2.1. Flexible Verbraucher

Unter flexiblen Verbrauchern versteht man die Verbraucher, deren Strombedarf oder ein Teil davon in zeitlichen Grenzen variieren kann. Somit haben sie die Möglichkeit, ihren Leistungsbedarf zu einem möglichst günstigen Zeitpunkt befriedigen zu können.

4.2.2. Unflexible Verbraucher

Unflexible Verbraucher passen ihren Energiebedarf nicht an die sich verändernden Strompreise an. Da davon ausgegangen wird, dass wegen der Kostenersparnis unflexible Verbraucher durch flexible ersetzt werden, reicht es, wie in Abschnitt 4.2.3 beschrieben, unflexible Verbraucher durch flexible zu modellieren.

4.2.3. Lastprofile

Zur Beschreibung des Verhaltens der Verbraucher im Verlauf eines Tages (Lastkurve) werden mehrere Profile verwendet. Ein Profil für den *unflexiblen Anteil*, das die *prognostizierte und die tatsächlich benötigte Leistung* in Form zweier Polynome (dargestellt durch Tabellen) angibt. Dazu ein Profil für den *flexiblen Anteil*, bestehend aus den folgenden Daten:

- frühester Lieferzeitpunkt
- spätester Lieferzeitpunkt
- Dauer des Leistungsbedarfs
- Leistung (Polynom, z. B. als Tabelle)
- Anzahl der Lieferungen
- max. Preis

Die ersten Parameter legen den gewünschten *Lieferzeitraum* fest. Im Fall eines flexiblen Verbrauchers darf die Stromlieferung zu einem beliebigen Zeitpunkt innerhalb des Intervalls zwischen frühestem und spätestem Lieferzeitpunkt stattfinden. Die *Dauer des Leistungsbedarfs* muss für *flexible Verbraucher* kleiner dem o. g. Intervall sein. Wenn der Lieferzeitraum und

das Intervall gleich lang sind, ist der Lieferzeitpunkt nicht mehr variabel. Auf diese Weise lassen sich *unflexible Verbraucher* modellieren.

Die gewünschte *Leistung* wird in Form eines Polynoms (Lastprofil) übergeben. Das Lastprofil kann innerhalb des Lieferzeitraums mehrfach angefordert werden (*Anzahl der Lieferungen*). Der *Preis*, den der Verbraucher maximal zu zahlen bereit ist, wird übergeben. Ein unendlich hoher Preis steht dafür, dass die Leistung unbedingt notwendig ist, und niedrigere Preise bedeuten, dass der Verbraucher gegebenenfalls auf die Leistung verzichten kann. (Der flexibel bezogene Teil der Leistung wird wie verhandelt abgenommen.)

4.3. Möglicher Preisrahmen

Der Zukaufspreis $p_z(t)$ vom Verbundnetz und der Verkaufspreis $p_v(t)$ an das Verbundnetz für den Bilanzkreis sind extern über die Zeit gegeben. Aus diesen Preisen wird der Bilanzkreis seinen Angebotspreis $p_a(t)$, zu dem er Strom anbietet, und seinen Kaufpreis $p_k(t)$, zu dem er Strom aus seinem Netz bezieht, bestimmen und veröffentlichen. Diese Preise werden sich aufgrund ihrer Abhängigkeit vom Preis des Verbundnetzes ebenfalls mit der Zeit ändern. Dabei gilt, dass $p_a(t)$ über $p_z(t)$ liegt, da der Bilanzkreis Strom, den er aus dem Verbundnetz bezieht, zu einem höheren Preis weiterverkaufen möchte, um Gewinn zu machen.

Desweiteren liegt es auf der Hand, dass der Bilanzkreisverantwortliche weniger für Strom, den er aus dem eigenen Netz bezieht, bezahlt, als er vom Verbundnetz bekommt, also ist $p_k(t)$ kleiner als $p_v(t)$. Da der jeweilige Verkaufspreis des Verbundnetzes bzw. des Bilanzkreises höher sein wird als ihre Kaufpreise, gilt insgesamt also: [1]

$$p_a(t) \geq p_z(t) \geq p_v(t) \geq p_k(t) \geq 0$$

Daraus ergeben sich Konsequenzen für die Erzeuger-Agenten, die sich jederzeit über die aktuellen (und vergangenen) Preise des Bilanzkreises informieren können und u. U. sogar eine Prognose über die zukünftige Preisentwicklung erhalten. Die Erzeuger-Agenten müssen also, um erfolgreich handeln zu können, ihre Preise zwischen $p_k(t)$ und $p_a(t)$ setzen. Liegen sie

[1] Ebenso wäre es möglich, p_a und p_k direkt extern vorzugeben.

über $p_a(t)$ des Bilanzkreises, so wäre jeder Käufer gut beraten, beim Bilanzkreis zu kaufen. Sollten die Erzeuger auf diesem Preis verharren, so können sie letztlich ihren Strom nur zum $p_k(t)$ des Bilanzkreises an eben diesen verkaufen. Ein Angebotspreis unterhalb von $p_k(t)$ wäre ebenso sinnlos, da die Erzeuger ihren Strom dann besser direkt an den Bilanzkreis zu $p_k(t)$ verkaufen könnten. Daraus folgt:

$$p_a(t) \geq p_e(t) \geq p_k(t) \geq 0$$

Das Verhältnis von Preisen des Verbundnetzes und der Erzeuger ist unerheblich, da das Verbundnetz nie direkt, sondern immer nur indirekt über den Bilanzkreis (BK) in Erscheinung tritt.

Hier ein einfaches Beispiel: $p_a(t) = 2\,€/\text{kWh}$, $p_k(t) = 1\,€/\text{kWh}$

1. $p_e(t) > 2\,€/\text{kWh}$: Die Käufer würden günstiger beim BK kaufen. Der Anbieter könnte also nicht verkaufen.

2. $p_e(t) < 1\,€/\text{kWh}$: Der Anbieter könnte zu $p_k(t) = 1\,€/\text{kWh}$ an den BK besser verkaufen.

3. $2\,€/\text{kWh} \geq p_e(t) \geq 1\,€/\text{kWh}$: Bei diesem Preisintervall realisieren sowohl der Anbieter als auch der Käufer einen finanziellen Vorteil.

4.4. Blackboard und Vertragsabschluss

Dem Blackboard-Agenten werden von den Erzeuger-Agenten Angebote geschickt, die den Verbraucher-Agenten zugänglich gemacht werden. Ein Angebot besteht aus der *angebotenen Leistung*, dem *Preis*, dem *Zeitraum der Lieferung* und der *Dauer der Gültigkeit* des Angebotes. Die Angebote werden genau dann zurückgezogen und gelöscht, wenn der Gültigkeitszeitraum abläuft oder das Angebot von einem Verbraucher-Agenten angenommen wurde.

Die Verbraucher-Agenten fordern vom Blackboard Angebote an, die bestimmten Bedingungen (Preis, Zeitraum, Leistung) genügen. Das Blackboard schickt den Agenten Listen mit den entsprechenden Angeboten. Bis zum Ablauf des angefragten Zeitraums informiert das Blackbaord die Agenten selbständig über neue Angebote.

Aus den passenden Angeboten können die Verbraucher-Agenten ihren Vorstellungen entsprechend auswählen. Sie stellen Kontakt zum zuständigen Erzeuger-Agenten her und übermitteln ihr eigenes Angebot. Dieses kann sowohl im Lieferzeitraum als auch in der Leistung, jedoch nicht im Preis variieren. Der Preis ist somit nicht verhandelbar. Dem Erzeuger steht es in diesem Fall frei, das Angebot des Verbrauchers anzunehmen oder abzulehnen, jedoch kann nicht unmittelbar weiter verhandelt werden.

Nimmt ein Verbraucher das Angebot eines Erzeugers vollständig in Anspruch, d. h. weder Lieferzeitraum noch gewünschte Leistung weichen vom Angebot des Erzeugers ab, so darf der Erzeuger den Vertrag nur dann ablehnen, wenn bereits ein anderer Verbraucher den Zuschlag erhalten hat.

Lehnt der Erzeuger ein Angebot eines Verbrauchers ab, kommt kein Vertrag zustande, der Verbaucher kann allerdings sein Glück zu einem späteren Zeitpunkt erneut versuchen. Wird ein Vertrag über nur einen Teil des ursprünglichen Angebots des Erzeugers abgeschlossen, so löscht der Erzeuger-Agent das alte Angebot und stellt entsprechend ein oder mehrere neue Angebote über den verbleibenden Rest an das Blackboard.

4.5. Verbundnetz und Bilanzkreisverantwortlicher

Das Verbundnetz (VN) sichert die Stromversorgung im Bilanzkreis dadurch, dass der Bilanzkreis jederzeit zu einem Zukaufspreis $p_z(t)$ Leistung aus dem VN zukaufen kann, um sein eigenes Netz auszubalancieren. Ebenso besteht für den Bilanzkreisverantwortlichen bei einem Leistungsüberschuss die Möglichkeit, überschüssige Leistung an das VN zum Verkaufspreis $p_v(t)$ zu verkaufen. Beide Preise sind extern gegeben, ändern sich aber über den Tagesverlauf. Sie sind als Funktion in Abhängigkeit von der Zeit gegeben. Sie bestimmen das Preisniveau, mit dem der Bilanzkreis Strom anbietet bzw. kauft (siehe Abschnitt 4.3).

Eine weitere Aufgabe des Bilanzkreisverantwortlichen ist die Abrechnung der erfüllten Verträge zwischen Erzeugern und Verbrauchern. Ebenso sind – bei Abweichungen und nicht ausreichendem Angebot oder Nachfrage – Zukäufe und Verkäufe vom bzw. in das Verbundnetz abzurechnen. Hierzu

müssen die jeweiligen Preise, die vereinbarten und die tatsächlichen Leistungsflüsse bekannt sein. Folgende Fälle sind möglich:

1. *Vertraglich vereinbarte Leistung P_{soll} und tatsächliche gelieferte Leistung P_{ist} stimmen überein:* Verbraucher zahlt vereinbarten Preis $p \cdot P$ an Erzeuger.

2. *Erzeuger konnte nur weniger erzeugen als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Erzeuger zahlt an den Bilanzkreis für zugekaufte Leistung $p_a \cdot (P_{\text{soll}} - P_{\text{ist}})$.

3. *Erzeuger produziert mehr Leistung als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Erzeuger erhält für zusätzlich eingespeiste Leistung $p_k \cdot (P_{\text{ist}} - P_{\text{soll}})$ vom Bilanzkreis.

4. *Verbraucher benötigt mehr Leistung als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Verbraucher zahlt für zusätzliche Leistung an den Bilanzkreis $p_a \cdot (P_{\text{ist}} - P_{\text{soll}})$.

5. *Verbraucher benötigt weniger Leistung als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Verbraucher erhält für quasi eingespeiste Leistung $p_k \cdot (P_{\text{soll}} - P_{\text{ist}})$.

6. *Verbaucher bezieht Strom ohne Vertrag:* entspricht Fall 4 mit $P_{\text{soll}} = 0$.

7. *Erzeuger produziert Strom ohne Vertrag:* entspricht Fall 3 mit $P_{\text{soll}} = 0$.

Beide Funktionen (Abrechnung und Repräsentanz des Verbundnetzes) können in einem Agenten realisiert werden.

4.6. E/A-Schnittstellen

4.6.1. Eingabedaten

Die Eingabedaten für unsere Simulation sind Verbrauchsprofile, Wetterprofile, Brennstoffkosten, Anzahl der Agenten und die Preise für den Strom aus dem Verbundnetz.

Die *Verbrauchsprofile* (Lastvorschauen) werden in Form von Konfigurationsdateien übergeben. In diesen Dateien steht der prognostizierte Verbrauch und der reale Verbrauch. Diese Verbrauchsdaten sind zunächst willkürlich von uns gewählt.

Die *Wetterprofile* (Wetterprognoen) sind unterteilt in zwei Kategorien: Profile für die Lichtstärke und Profile für die Windgeschwindigkeit. Diese werden ebenfalls über Konfigurationsdateien übergeben und zwar zweifach, sowohl für die Prognose als auch für den realen Wetterverlauf. Die Wetterprofile entsprechen in ihrer Charakteristik realen Wetterdaten.

Die *Brennstoffkosten* und die Preise für den *Strom aus dem Verbundnetz* müssen so gewählt werden, dass die Brennstoffzelle konkurrenzfähig ist, d. h. es darf nicht passieren, dass die Energie, die die Brennstoffzelle produziert, so teuer ist, dass die Energie vom Verbundnetz (über den Bilanzkreis) billiger zur Verfügung gestellt werden kann.

4.6.2. Ausgabedaten

Zur Analyse der Simulation sind ausreichende Logfiles zur Verfügung zu stellen.

Um die Strategien der Erzeuger vergleichen zu können, ist neben dem zeitlichen Verlauf der von den simulierten Erzeugern tatsächlich erzeugten Leistung auch die Prognose, die der Erzeuger ermittelt, anzugeben, sowie die letztlich verkaufte Leistung (jeweils im zeitlichen Verlauf). So können unter Beibehaltung der Rahmendaten verschiedene Strategien zum Verkauf der erzeugten Leistung verglichen werden.

Analog sind für den Verbraucher der prognostizierte und reale Leistungsbedarf der Technik sowie die letztlich gekaufte Leistung im zeitlichen Verlauf auszugeben.

Um einen Überblick über die Gesamtsituation zu bekommen, sind vom Verbundnetz bezogene, ans Verbundnetz gelieferte Leistungen sowie deren Differenz, ebenso im zeitlichen Verlauf, auszugeben.

Die abgeschlossenen Verträge und die abgelehnten Vertragsanbahnungen sind mit relevaten Daten (Preis, Leistung, Zeitpunkt, Dauer) zu protokollieren.

Die Daten sollen in lesbarer Form in Logfiles und (zusammen mit den Profilen) in einer Datenbank gespeichert werden.

5. Pflichtenheft

Die bereits genannten Anforderungen sind zu erfüllen. Darüber hinaus sollen alle Module mittels TCP/IP kommunizieren, unabhängig davon, ob die Simulation auf einem oder mehreren Rechnern abläuft.

Zudem sollen alle Module ihre wichtigen Aktionen protokollieren. Dies betrifft insbesondere das Erzeuger-Modul und das dazugehörige Technik-Modul, das Verbraucher-Modul und das Bilanzkreis-Modul. Daten sollen dabei auch redundant gesammelt werden, d. h. auch wenn z. B. Daten des Bilanzkreis-Moduls aus den beiden anderen bestimmt werden könnten, soll das Modul seine Daten dennoch protokollieren.

Für diese erste Version ist das Gesamtsystem aus einem Bilanzkreis und nur einer geringen Zahl von Erzeugern und Verbrauchern aufzubauen, jedoch mind. drei Erzeugern je Typ, d. h. je mind. drei Brennstoffzellen, Windkraftanlagen und Solarzellen, und zudem mind. zehn Verbrauchern.

Darüber hinaus sind Testdaten für die Simulation sowie zu erwartende Ergebnisse festzulegen. Dies gilt besonders für die Preissituation, d. h. die Preise des Bilanzkreises einerseits sowie der Preise, zu denen Verbraucher letztendlich ihren Strom beziehen.

Auf jeden Fall müssen Verhandlungen stattfinden, es darf also nicht die Situation eintreten, dass alle Verbraucher ihren Strom vom Bilanzkreis beziehen.

Für die Auswertung der Simulation ist eine offline Analyse nach Ablauf des Experimentes aus den gespeicherten Protokolldaten ausreichend. Eine Echtzeit-Auswertung während der laufenden Simulation ist nicht erforderlich.

6. Design und Implementation

6.1. Programmiersprache und Entwicklungsumgebung

Zur Implementierung des Multiagentensystems standen die Sprachen C, C++ und Java sowie deren Kombination zur Wahl. Da die Simulation in der ersten Entwicklungsphase keine harten Echtzeitanforderungen (nur Firm Deadlines) stellt und die kleinste Abrechnungs- und Verhandlungseinheit eine Minute beträgt, waren an die Entwicklungssprache diesbezüglich keine besonderen Anforderungen zu stellen.

Aufgrund der größeren Erfahrung der Gruppe mit Java und der Meinung vieler Gruppenmitglieder, dass Java vor allem aufgrund von strikteren Prüfungen und hilfreicheren Meldungen des Compilers eine schnellere Entwicklung ermögliche, wurde als Implementationssprache Java (Sun Java 2 SE 1.4.2 für Linux) gewählt.

Für eine objektorientierte Programmiersprache kommt eine Vielzahl von Entwicklungsumgebungen in Frage, z. B. Together, Eclipse oder Rose. Auf Empfehlung des Software-Technologie-Labors (STL) des Fachbereichs wurde Eclipse gewählt. Gründe waren auch die besseren Supportmöglichkeiten durch das STL als für andere Umgebungen und die unsicheren Lizenzbedingungen für Together, dem Tool, mit dem die Gruppenmitglieder aus anderen Lehrveranstaltungen vertraut waren.

Zusätzlich wurde für die UML-Modellierung ein Eclipse-Plugin der Firma Omondo[1] ausgewählt.

6.2. Klassen-Struktur

Auf der Basis der Anforderungsanalyse, siehe Abschnitt 4 auf Seite 33, wurde ein objektorientiertes Design gewählt. Da Java als Implementie-

[1] Siehe Anhang, Seite 193

rungssprache gewählt wurde, sind die verschiedenen Agenten des Systems in eigenen Java Packages gekapselt, wobei die Schnittstellen zur Technik jeweils Teil der entsprechenden Agenten-Pakete sind und die Verbundnetz-Schnittstelle mit zum Bilanzkreis-Paket gehört. Hinzu kamen noch Pakete mit Hilfsklassen, die u. a. die Netzwerkkommunikation kapseln, sowie ein Paket zur Simulationssteuerung. Insgesamt gibt es damit die folgenden acht Pakete:

- DezentenLib
- DezentenLib.Log
- DezentenLib.Net
- BalanceDistrictAgent
- BlackBoardAgent
- ConsumerAgent
- ProducerAgent
- Simulation

Die Pakete werden im Zwischenbericht der Projektgruppe Dezenten [FHI+03b] detailliert beschrieben.

7. Integration, Simulationsläufe und Auswertung

7.1. Testszenario Eins

Der erste Test ist recht einfach gehalten, um das generelle Funktionieren des Systems zu demonstrieren. Angebotene und verbrauchte Energie sollen sich genau ausgleichen. Prognose und realer Verbrauch sind also identisch. Vom Bilanzkreis sollte keine Energie geliefert werden.

Um leichter nachvollziehbare Ergebnisse zu erhalten, nimmt nur eine Brennstoffzelle teil, da Brennstoffzellen wegen ihrer Wetterunabhängigkeit konstante Leistung liefern.

Am Test nehmen die folgenden Programmodule teil:

- Ein Energieerzeuger in Form einer Brennstoffzelle, welche genau 50 Kilowatt liefert. [1]
- Zwei Verbraucher, deren Profile insgesamt einen ständigen Verbrauch von 50 Kilowatt ergeben, siehe Abbildung 7.1.
- BlackBoard, Bilanzkreis und Controller als grundlegende Infrastruktur und zur Steuerung.

7.1.1. Erwartungen

- Beide Verbraucher können ihren Energiebedarf zu 100 % decken.
- Die gesamte Energie des Erzeugers ist aufgekauft, da die erzeugte Energie gleich der nachgefragten Energie ist.

[1] Eine Abweichung zwischen Prognose und realen Werten kann also im Gegensatz zu wetterabhängigen Erzeugern, Solarzelle und Windkraft, nicht auftreten.

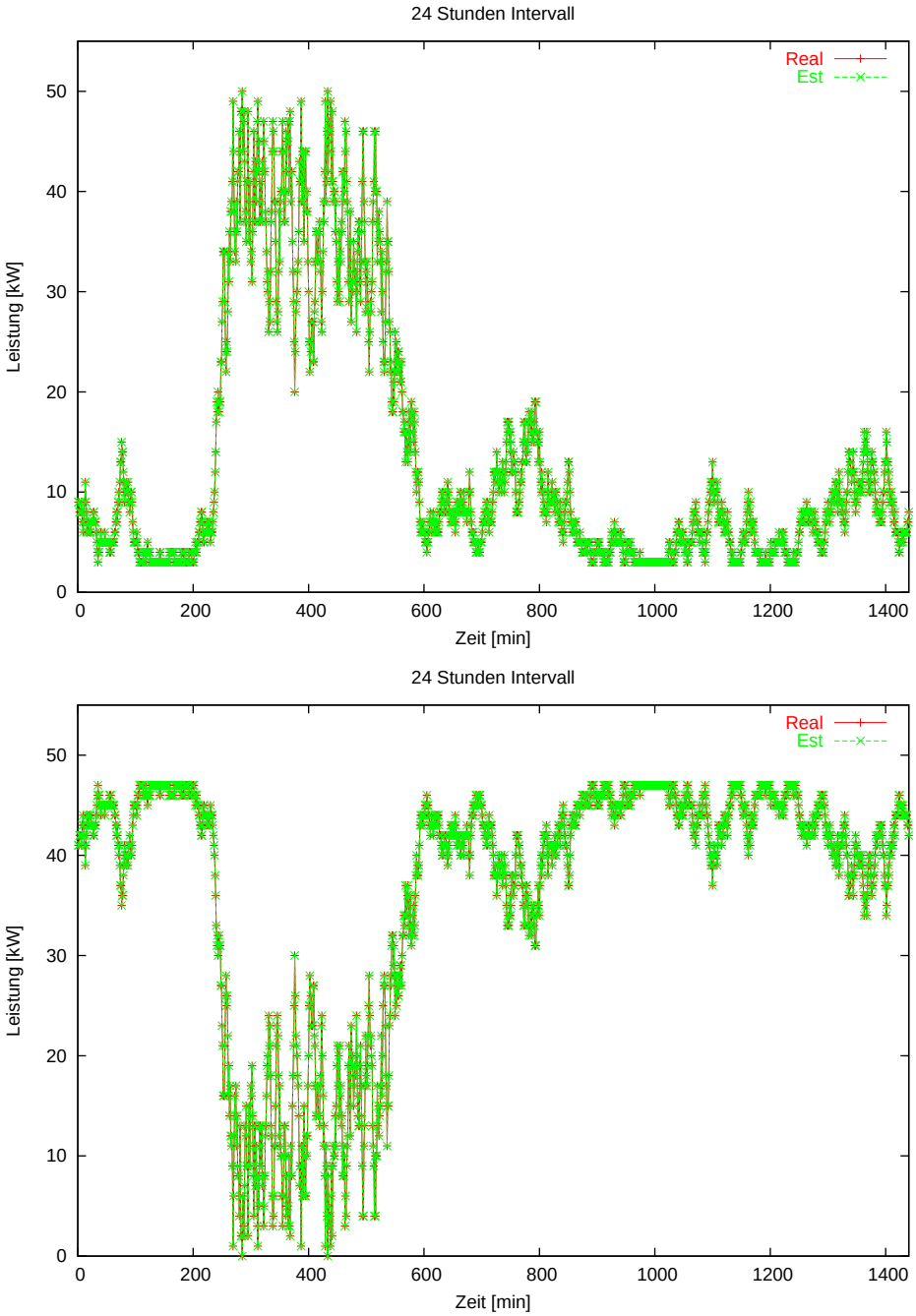

Abbildung 7.1.: Die Profile der 2 Verbraucher.

- Am Ende der Verhandlungen sollen keine Angebote mehr am Blackboard stehen.
- Es wird nichts vom Bilanzkreis eingekauft.

7.1.2. Beobachtungen

- Der Energiebedarf der Verbraucher konnte in diesem Testszenario durch die Leistung der Brennstoffzelle vollständig gedeckt werden.
- Die vom Erzeuger angebotene Energie wurde vollständig aufgekauft.
- Nach der Deckung des Gesamtenergiebedarfs waren keine Angebote mehr auf dem Blackboard vorhanden.
- Der gesamte Energiebedarf wurde ausschließlich durch die Erzeuger gedeckt, ohne Hinzunahme des Bilanzkreises.
- Es wurde kein Überschuss produziert.

7.2. Testszenario Zwei

In diesem Test soll ein komplexeres Szenario aufgebaut werden, um das Zusammenspiel der Komponenten zu untersuchen. Hierzu nehmen zwei Verbraucher und insgesamt vier Erzeuger an dem Test teil:

- Die Verbraucher sind auf einen Bedarf von jeweils 75 Kilowatt eingestellt, mit einem Peak von je 150 Kilowatt von 12:00 bis 13:00 Uhr und einem Bedarfsloch von je 0 Kilowatt von 18:00 bis 19:00 Uhr, siehe Abbildung 7.2.
- Die Erzeuger setzen sich aus zwei Brennstoffzellen zu 40 bzw. 60 kW, einer Windkraftanlage und einer Solarzelle zusammen.
- In diesem Szenario weichen Wetterprognose und reales Wetter voneinander ab.
- Zusätzlich nehmen Blackboard, Bilanzkreis und der Controller teil.

- In diesem Szenario haben wir auf die geforderte Anzahl von jeweils drei Erzeugern, d.h. drei Solarzellen, drei Windkrafträdern und drei Brennstoffzellen verzichtet, da die Möglichkeiten zur Auswertung dieser großen Datenmengen fehlten und haben uns auf obige Konstellation beschränkt (ein nicht ausgewerteter Test mit 40 Erzeugern funktionierte bis auf die Auswertung allerdings problemlos).

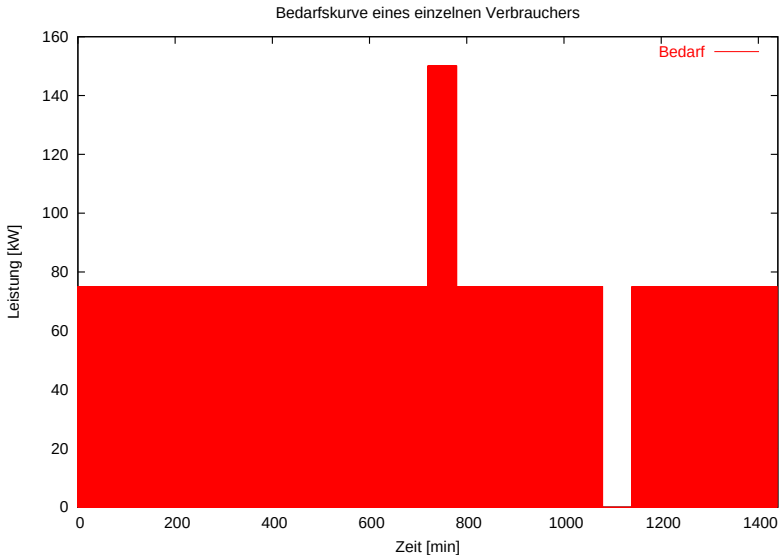

Abbildung 7.2.: Jeweiliger Bedarf der Verbraucher 1 und 2 im Testszenario 2

7.2.1. Erwartungen

- Es ist zu erwarten, dass die Erzeuger, bis auf die Zeit von 12:00 bis 13:00 Uhr, den Bedarf der Verbraucher decken können.
- Es soll zumindest in den beiden Extremsituationen der Bilanzkreis einspringen, um zu viel produzierte Energie von den Erzeugern abzunehmen bzw. um mit zusätzlicher Energie auszuhelfen.
- Die Agenten sollen problemlos über die geforderte Socket-Kommunikation in einem verteilten System interagieren können.

7.2.2. Beobachtungen

- Die Bedarfsdeckung der Verbraucher war problemlos möglich, jedoch konnte in der Zeit von 12:00 bis 13:00 die Nachfrage erwartungsgemäß nicht vollständig gedeckt werden (siehe Abbildungen 7.3, 7.8 sowie 7.9).
- Erwartungsgemäß weichen bei Windkraftanlagen und Solaranlagen die Kurven von erzeugter und verkaufter Energie voneinander ab (siehe Abbildungen 7.5 und 7.6), während dies bei den Brennstoffzellen nicht passiert. Letztere liefern jedoch nur Energie, wenn tatsächlich Bedarf besteht (siehe Abbildungen 7.4 und 7.7).
- Der Bilanzkreis deckte die entsprechenden Defizite aus dem Verbundnetz bzw. kaufte die Überschüsse zur Einspeisung in das Verbundnetz auf.
- Die verteilte Kommunikation via TCP/IP funktionierte fehlerfrei.

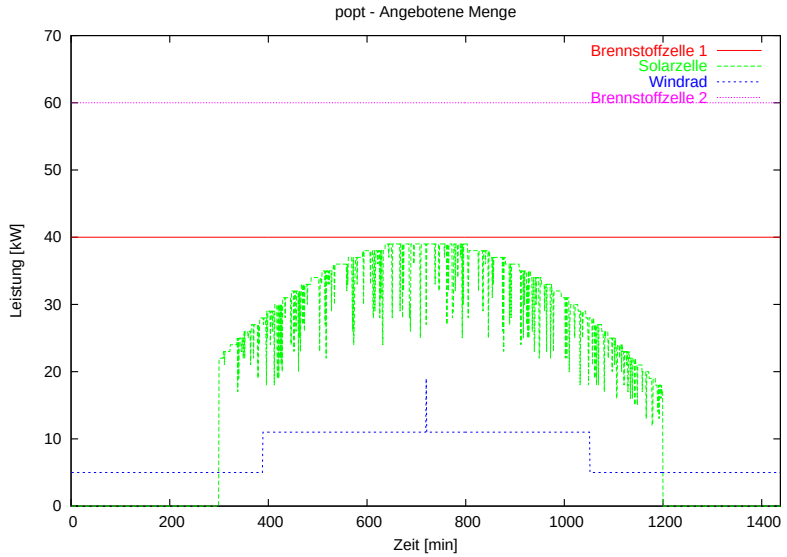

Abbildung 7.3.: Angebotene Leistung

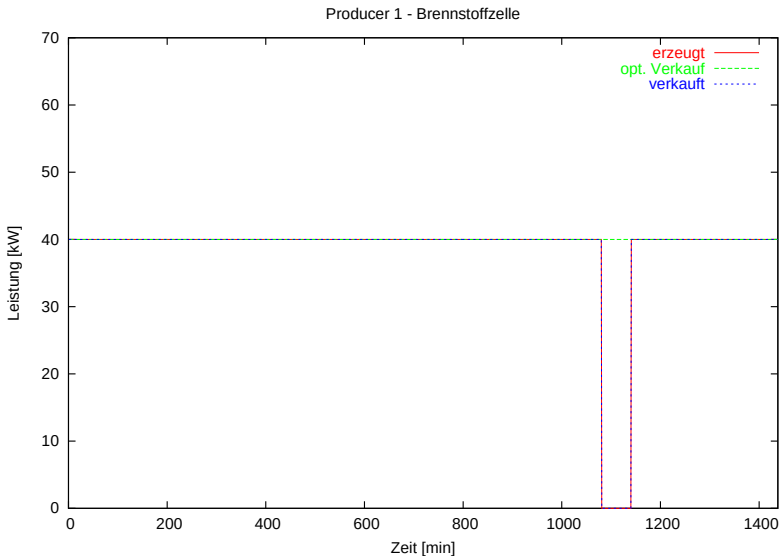

Abbildung 7.4.: Erzeuger 1: Brennstoffzelle

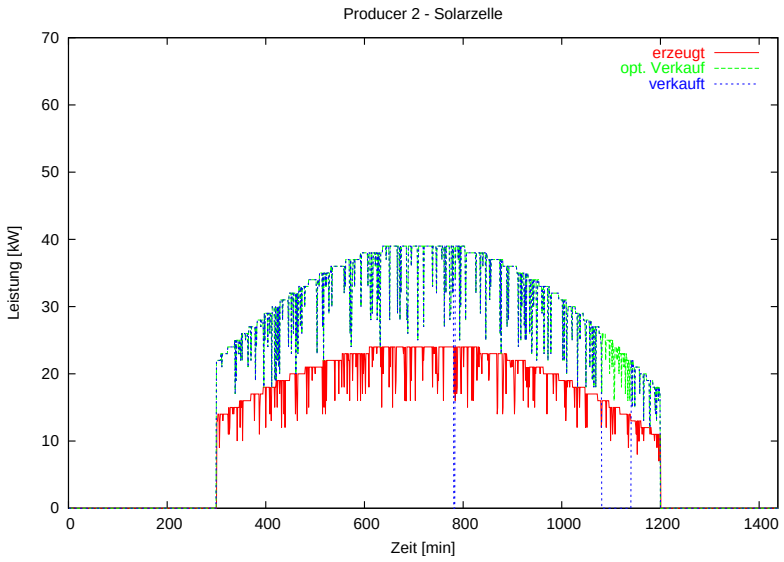

Abbildung 7.5.: Erzeuger 2: Solarzelle

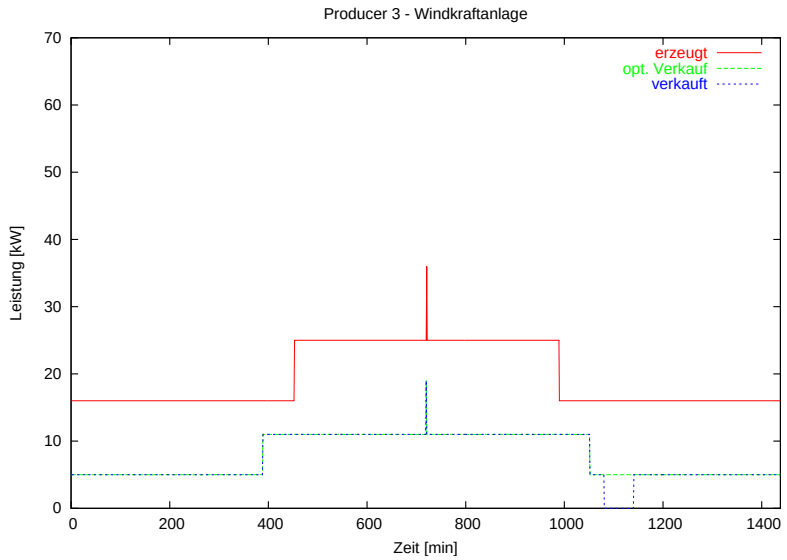

Abbildung 7.6.: Erzeuger 3: Windkraftanlage

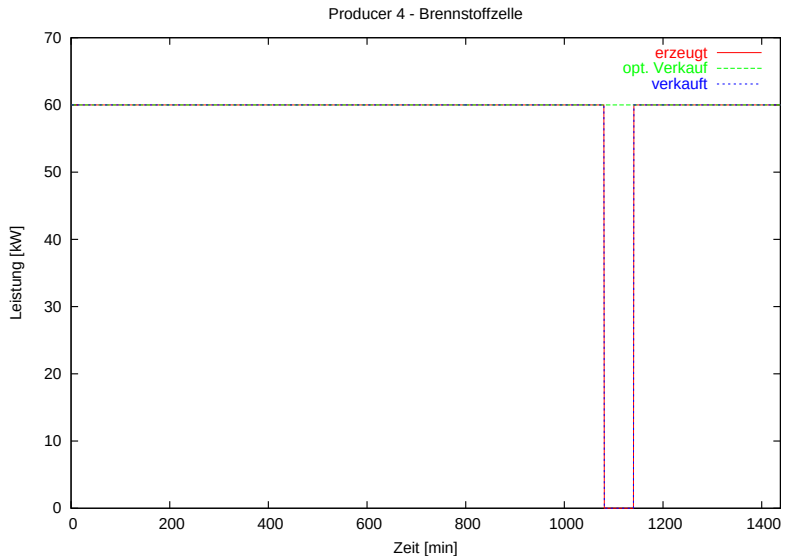

Abbildung 7.7.: Erzeuger 4: Brennstoffzelle

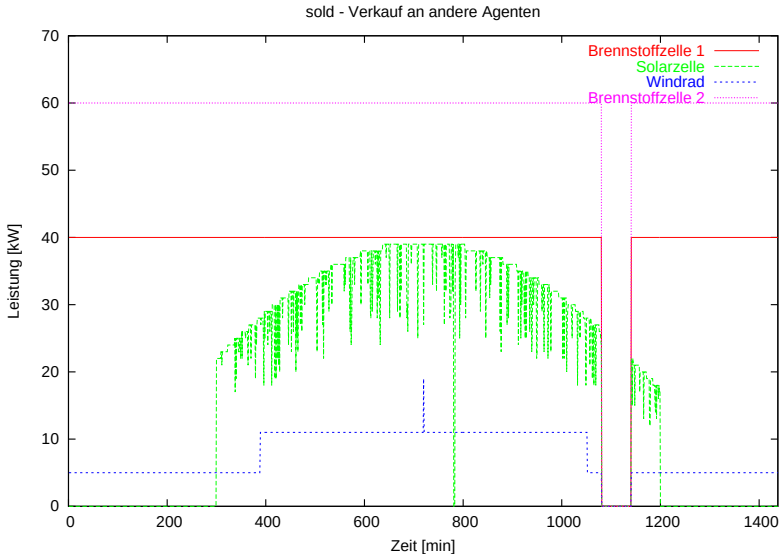

Abbildung 7.8.: Vertraglich vereinbarte Lieferungen

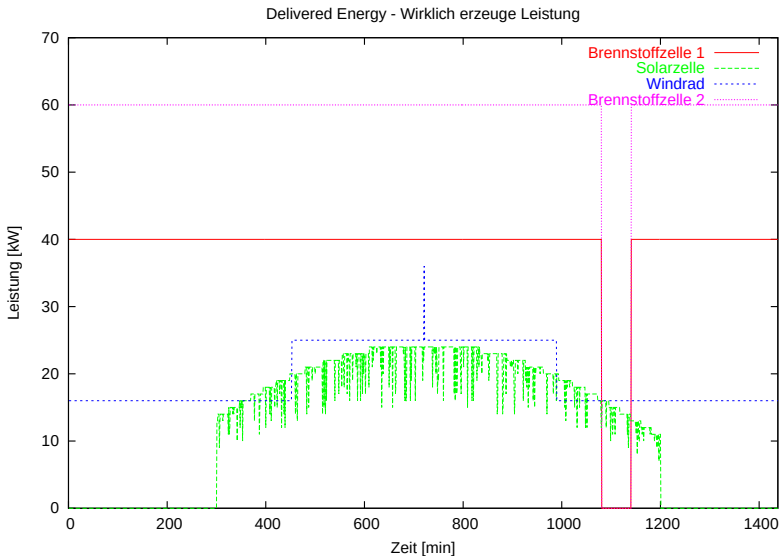

Abbildung 7.9.: Tatsächlich erzeugte Leistung

7.3. Ergebnisse des ersten Semesters

Das erste Testszenario sollte zeigen, dass die grundlegenden Funktionen der Simulation keine Fehler aufweisen. Die Kommunikation zwischen den Agenten, Erstellung von Angeboten und Verhandlung zwischen Verbrauchern und Produzenten fanden tatsächlich statt. Ebenso wurde gezeigt, dass Verbraucher tatsächlich bei Produzenten einkaufen und nicht ihren Bedarf komplett beim Bilanzkreis zu decken versuchen.

Aus dem zweiten Testszenario geht hervor, dass ein verteilter Ablauf der Simulation auf mehreren Rechnern und die Mischung verschiedener Produzententypen funktioniert, so dass Verbraucher ihren Bedarf soweit wie möglich bei den Produzenten statt beim Bilanzkreis decken.

Damit ist die grundlegende Funktionalität der Simulation sichergestellt. Eine genauere Analyse der Verhandlungsstrategien wurde für die zweite Phase der Projektgruppe vorgesehen, wie auch im Zeitplan (siehe Abschnitt 2.5) beschrieben.

Ein zusätzliches Ergebnis des zweiten Testszenarios ist die Erkenntnis, dass bei komplexeren Simulationen mit vielen Produzenten und Verbrauchern eine enorme Datenmenge anfällt, die ohne passende Hilfswerkzeuge kaum sinnvoll auszuwerten ist. Da diese Werkzeuge für die erste Phase nicht vorgesehen waren, musste bei komplexeren Simulationsläufen, wie im Pflichtenheft vorgesehen, auf eine Auswertung verzichtet werden. In der zweiten Phase sollten also entsprechende Auswertungwerkzeuge von Beginn an mit eingeplant werden, zumal nun leichter die nötigen Anforderungen an diese formuliert werden können.

Zusammenfassend bleibt festzustellen, dass die Projektgruppen-Arbeit ohne nennenswerte Probleme verlaufen ist. Die Zusammenarbeit der Teilnehmer verlief zufriedenstellend, Konflike wurden von der Gruppe ohne Hilfe durch die Betreuer schnell und produktiv gelöst. Durch unterschiedliche Nebenfächer und Interessen konnten sich die Teilnehmer gut ergänzen.

Kritisierend anzumerken ist der unterschätzte Aufwand für die Integration der Module und die Gesamttests sowie für die Endversion des Zwischenberichts. Durch den Verzug konnten nur die in Kapitel 7 beschriebenen Tests durchgeführt werden. Das intensive Experimentieren mit der vorliegenden Version musste auf das zweite Semester verschoben werden.

Teil III.

Konzeption und Implementation – Zweite Phase

In der zweiten Phase von DEZENTEN wurden auf dem Wissensstand der ersten Version zahlreiche Verbesserungen und Erweiterungen in das bestehende Programm eingebaut. Dabei konnte auf dem Code, der im ersten Semester programmiert wurde, aufgebaut werden. Ein anderer Teil musste ersetzt und stark erweitert werden. Beispielhaft für die Erweiterungen sind die Erhöhung der Benutzerfreundlichkeit durch eine graphische Benutzeroberfläche, erhöhte Adaptivität der Agenten, um auf kurzfristige Änderungen zu reagieren, sowie ausführlichere Tests zu nennen.

Im Wesentlichen hinzugekommen oder verbessert wurden:

- Behandlung von Leitungsausfällen
- Nachverhandlungen der Agenten
- graphische Benutzeroberfläche
- Prognosekorrektur

8. Anforderungshandbuch

In diesem Teil werden die Anforderungen an die zweite Version von DEZENTEN beschrieben. Das System soll durch Agenten realisiert werden. Für die Agenten wird die nachfolgende Funktionalität gefordert. Zur besseren Wartbarkeit und Erweiterbarkeit soll das System modular aufgebaut werden. Jedes Modul soll durch mindestens einen selbständigen Agenten realisiert werden. Die Agenten sind im Einzelnen:

- Erzeuger-Agent, Abschnitt 8.1
- Verbraucher-Agent, Abschnitt 8.2
- Agent für das Verbundnetz (VN) und den Bilanzkreisverantwortlichen (BKV), Abschnitt 8.5
- Blackboard-Agent (BB), Abschnitt 8.6

Die Struktur und Kooperationsbeziehungen des Multiagenten-Systems sind in Abbildung 8.1 auf der nächsten Seite wiedergegeben. Sämtliche zeitbezogenen Berechnungen werden mit einer Auflösung von einer Minute durchgeführt.

8.1. Erzeuger-Agent und Technik

Die Erzeuger-Agenten repräsentieren die Interessen der Betreiber dezentraler Energieerzeugungsanlagen. Die möglichen Anlagen sind:

- Brennstoffzellen
- Solaranlagen
- Windkraftanlagen

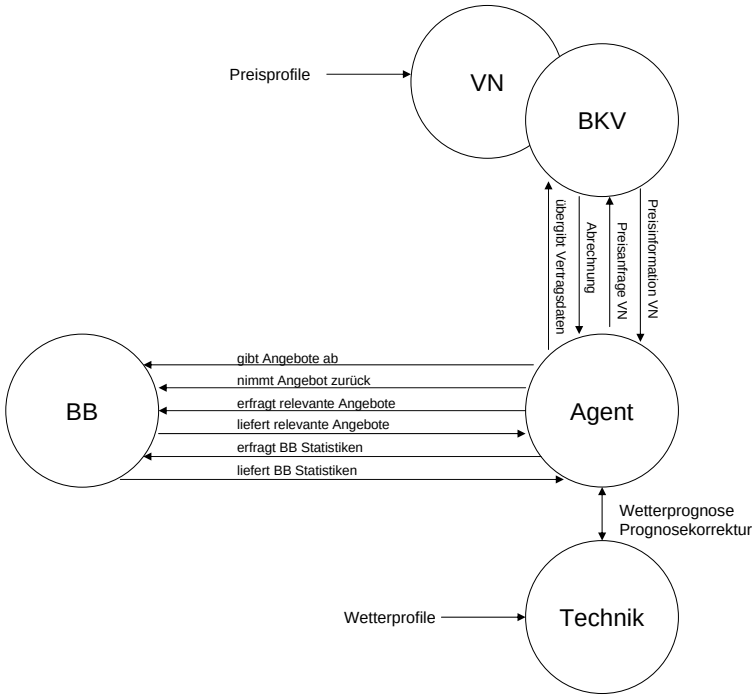

Abbildung 8.1.: Struktur des Multiagenten-Systems

Ziel der Erzeuger-Agenten ist, Angebote über die Lieferung elektrischer Energie zu erstellen und sie Verbrauchern elektrischer Energie über ein Blackboard zugänglich zu machen. Diese Angebote können von den Verbrauchern zeitlich oder leistungs-gesplittet angenommen werden (zum Ablauf der Verhandlung siehe Abschnitt 8.4).

Brennstoffzellen: Liegt eine Brennstoffzelle vor, so kann diese prinzipiell jederzeit Leistung erzeugen. Für den Erzeuger-Agenten ist nun entscheidend, ob er die Leistung zu einem höheren Preis verkaufen kann, als Kosten zur Erzeugung (Brennstoffkosten, Wartung, ...) entstehen. Um also einen vernünftigen Verkaufspreis $p_e(t)$ bestimmen zu können, benötigt er vom Technik-Modul den zeitlichen Verlauf der Kosten, die entstehen, um Strom zu produzieren. Dies entspricht einem Diagramm, in dem die Leistung gegen die Kosten aufgetragen wird. Dieses Diagramm ist in unserem Modell tagesunabhängig, da die Brennstoffkosten als konstant angenommen werden. Sollte der Erzeuger-Agent mit diesen Daten einen Vertrag abschließen können, so gibt er die entsprechenden Daten (Zeitraum, Leistung) des Vertrages an das Technik-Modul weiter.

Wind-, Solarenergie: Da hier die Leistung nicht vorhersehbar schwankt und nicht permanent abgerufen werden kann, benötigt der Erzeuger-Agent andere Informationen von der Technik.

Zum Beginn einer Verhandlung braucht der Erzeuger-Agent den Zeitraum, in dem Leistung verkauft werden kann, sowie eine Angabe über die Leistung, die voraussichtlich produziert werden kann. Hierzu erhält er die Prognose vom Technik-Modul. Diese beschreibt die Leistung, die aufgrund aktueller Wettervorhersagen produziert werden kann. Aus dieser Prognose kann der Erzeuger-Agent daraufhin ein Angebot über eine berechnete Menge zu einem exakt kalkulierten Preis machen. Dabei kann das Risiko für ihn umso größer werden, je mehr Leistung er anbietet, da die Prognose vom realen Verlauf stark abweichen kann. Wiederum übergibt der Erzeuger-Agent die notwendigen Vertagsdaten an die Technik. Die Prognose wird aufgrund aktueller Entwicklung fortgeschrieben und kann sich ändern, wenn der Zeitpunkt, auf den sie sich bezieht, näher rückt. Falls die Prognose eine höhere Leistung in Aussicht stellt, kann der Erzeuger-Agent die

voraussichtlich zusätzlich verfügbare Menge zum vorhandenen Angebot am Blackboard hinzufügen. Falls die aktuelle Prognose einen niedrigeren Wert vorhersagt als die vorherige, dann soll der Erzeuger-Agent zunächst versuchen, sein noch nicht verkauftes Angebot am Blackboard zu reduzieren. Falls die Leistung seines Angebotes nicht mehr ausreicht, um die Differenz der Prognosen auszugleichen, hat der Erzeuger-Agent mehr Leistung verkauft, als er voraussichtlich produzieren wird. In diesem Fall wird er in der Funktion eines Verbraucher-Agenten tätig und versucht, die fehlende Leistung am Blackboard nachzukaufen, um sicherzustellen, dass die Leistung, die er verkauft hat, tatsächlich geliefert wird. Ansonsten müßte der Bilanzkreisverantwortliche (BKV) mit (teurerer Leistung) einspringen.

Wenn der Zeitpunkt der Erzeugung näher rückt, die Erzeuger-Agenten jedoch Angebote am Blackboard haben, die sie noch nicht verkauft haben, werden sie die Preise nach unten anpassen, wenn wirtschaftlich möglich, um damit zu erreichen, dass ihre Angebote bei Mindernachfrage gegenüber den Angeboten der Konkurrenten von den Verbraucheragenten bevorzugt werden.

Falls ein Leitungssegment oder die Technik des Erzeugers ausfallen sollte, muss er auf den Ausfall angemessen reagieren können. Unabhängig davon, ob nun ein Ausfall eines Leistungssegmentes oder der Technik vorliegt, muss der Erzeuger, sobald er diesen Ausfall festgestellt hat, seine Angebote überprüfen. Alle Angebote, die er noch nicht verkauft hat und die deshalb noch am Blackboard stehen, soll er löschen. Die Angebote, die er schon verkauft hat, wird er nicht liefern können. Um seinen Verlust zu minimieren, soll er genau die Menge, die er im Zeitintervall des Technik- bzw. Leitungsausfalls verkauft hat, bei anderen Erzeugern nachkaufen, damit diese an seiner Stelle den jeweiligen Verbraucher beliefern.

Zur Abrechnung muß die Technik die tatsächlich eingespeiste Leistung zum Vertragszeitpunkt an den Erzeuger-Agent leiten, damit die Abrechnung mit dem Verbraucher-Agent und Bilanzkreisverantwortlichem erfolgen kann, siehe Abschnitt 8.5.

8.2. Verbraucher-Agent und Verbraucher

Es gibt flexible und unflexible Verbraucher, deren Unterschiede in 8.2.1 und 8.2.2 beschrieben werden. Auf diese Weise können sowohl verschiedene Arten von Industrieunternehmen als auch die Gesamtheit aller Haushalte eines Bilanzkreises leicht modelliert werden. Einzelhaushalte spielen für diese Simulation nur in sofern eine Rolle, als dass die einzelnen Schwankungen im Energieverbrauch in der Masse untergehen, so dass alle Haushalte zu einem unflexiblen Verbraucher mit gut vorhersehbarem Lastprofil zusammengefasst werden können.

Die Verbraucher-Agenten erfahren von ihrem Verbraucher-Modul den prognostizierten Energiebedarf in Form eines Lastprofils (siehe Abschnitt 8.2.3), so dass bekannt ist, wann voraussichtlich wieviel Leistung benötigt wird. Anschließend versucht der Verbraucher-Agent, den Energiebedarf des simulierten Verbrauchers durch Vertragsabschlüsse mit einem oder mehreren Erzeugeragenten zu decken. Angebote werden über das Blackboard 8.6 bekannt gemacht.

Sollte auf diese Weise nicht der gesamte Energiebedarf des Verbrauchers durch dezentral organisierte Erzeuger gedeckt werden können, so wird die fehlende Menge aus dem Verbundnetz bezogen, das zur Energielieferung jederzeit und ohne explizite Vertragsabschlüsse zu Verfügung steht.

Nach Ablauf des Vertragszeitraums findet die *Abrechnung* statt. Der Verbraucher-Agent rechnet die vertragsgemäß gelieferte Energie mit dem entsprechenden Erzeuger-Agenten ab. Abweichungen von den vereinbarten Leistungen werden mit dem Bilanzkreis abgerechnet, siehe Abschnitt 8.5.

8.2.1. Flexible Verbraucher

Unter flexiblen Verbrauchern versteht man Verbraucher, die ihren Strombedarf ganz oder teilweise in zeitlichen Grenzen variieren können. Somit haben sie die Möglichkeit, ihren Leistungsbedarf zu einem möglichst günstigen Zeitpunkt befriedigen zu können.

8.2.2. Unflexible Verbraucher

Unflexible Verbraucher passen ihren Energiebedarf nicht an die sich verändernden Strompreise an. Da davon ausgegangen wird, dass wegen der Kos-

tenersparnis unflexible Verbraucher durch flexible ersetzt werden, reicht es, wie in Abschnitt 8.2.3 beschrieben, unflexible Verbraucher durch flexible Verbraucher ohne Flexibilität zu modellieren.

8.2.3. Lastprofile

Zur Beschreibung des Verhaltens der Verbraucher im Verlauf eines Tages (Lastkurve) werden mehrere Profile verwendet. Ein Profil für den *unflexiblen Anteil*, das die *prognostizierte und die tatsächlich benötigte Leistung* in Form zweier Polynome (dargestellt durch Tabellen) angibt. Zudem gibt es ein Profil für den *flexiblen Anteil*, bei dem ein zeitlich begrenzter Energiebedarf innerhalb eines bestimmten Intervalls befriedigt werden muß (Lieferzeitraum), bestehend aus den folgenden Daten:

- frühester Lieferzeitpunkt
- spätester Lieferzeitpunkt
- Dauer des Leistungsbedarfs
- Leistung (Polynom als Tabelle)
- Anzahl der Lieferungen
- max. Preis

Die ersten beiden Parameter legen den gewünschten *Lieferzeitraum* fest. Im Fall eines flexiblen Verbrauchers darf die Stromlieferung zu einem beliebigen Zeitpunkt innerhalb des Intervalls zwischen frühestem und spätestem Lieferzeitpunkt stattfinden. Die *Dauer des Leistungsbedarfs* muss für *flexible Verbraucher* kleiner als der mögliche Lieferzeitraum sein. Wenn der Lieferzeitraum und das Intervall gleich lang sind, ist der Lieferzeitpunkt nicht mehr variabel, auf diese Weise lassen sich *unflexible Verbraucher* modellieren.

Die gewünschte *Leistung* wird in Form eines Lastprofils (Polynom) übergeben. Das Lastprofil kann innerhalb des Lieferzeitraums mehrfach angefordert werden (*Anzahl der Lieferungen*), z.B. Kühlung, die innerhalb von 24 Stunden zweimal anspringen muß. Der *Preis*, den der Verbraucher maximal zu zahlen bereit ist, ist fest. Ein unendlich hoher Preis steht dafür,

dass die Leistung unbedingt notwendig ist, und niedrigere Preise sind möglich, wenn der Verbraucher gegebenenfalls auf die Leistung verzichten kann. (Der flexibel bezogene Teil der Leistung wird wie verhandelt abgenommen.)

8.2.4. Adaptivität

Nachdem der Verbraucher sein Lastprofil angefordert und dementsprechend eingekauft hat, ist es möglich, dass sich sein Leistungsbedarf kurzfristig aber unvorhersehbar ändert. Dies geschieht durch ein aktualisiertes Lastprofil, das von dem vorherigen abweicht, nachdem eingekauft wurde. Hier kann für einen oder mehrere Zeiträume ein geringerer oder höherer Leistungsbedarf bestehen. In erstem Fall verkauft der Verbraucher den zuviel gekauften Strom weiter, um den gemachten Verlust im Rahmen zu halten. Er tritt dann als Erzeuger in Aktion und stellt Angebote für seinen Strom an das Blackboard. Im zweiten Fall kauft er Strom nach.

8.3. Möglicher Preisrahmen

Der Zukaufspreis $p_z(t)$ vom Verbundnetz und der Verkaufspreis $p_v(t)$ an das Verbundnetz für den Bilanzkreis sind extern über die Zeit gegeben. Aus diesen Preisen wird der Bilanzkreis seinen Angebotspreis $p_a(t)$, zu dem er Strom anbietet, und seinen Kaufpreis $p_k(t)$, zu dem er Strom aus seinem Netz bezieht, berechnen und veröffentlichen. Diese Preise werden sich aufgrund ihrer Abhängigkeit vom Preis des Verbundnetzes ebenfalls mit der Zeit ändern. Dabei gilt, dass $p_a(t)$ über $p_z(t)$ liegt, da der Bilanzkreis Strom, den er aus dem Verbundnetz bezieht, zu einem höheren Preis weiterverkaufen möchte, um Gewinn zu machen.

Desweiteren liegt es auf der Hand, dass der Bilanzkreisverantwortliche weniger für Strom, den er aus dem eigenen Netz bezieht, bezahlt, als er vom Verbundnetz bekommt, also ist $p_k(t)$ kleiner als $p_v(t)$. Da der jeweilige Verkaufspreis des Verbundnetzes bzw. des Bilanzkreises höher sein wird als ihre Kaufpreise, gilt insgesamt also: [1]

$$p_a(t) \geq p_z(t) \geq p_v(t) \geq p_k(t) \geq 0$$

[1] Ebenso wäre es möglich, p_a und p_k direkt extern vorzugeben.

Daraus ergeben sich Konsequenzen für die Erzeuger-Agenten, die sich jederzeit über die aktuellen (und vergangenen) Preise des Bilanzkreises informieren können und u. U. sogar eine Prognose über die zukünftige Preisentwicklung erhalten. Die Erzeuger-Agenten müssen also, um erfolgreich handeln zu können, ihre Preise zwischen $p_k(t)$ und $p_a(t)$ setzen. Liegen sie über $p_a(t)$ des Bilanzkreises, so wäre jeder Käufer gut beraten, beim Bilanzkreis zu kaufen. Sollten die Erzeuger auf diesem Preis verharren, so können sie letztlich ihren Strom nur zum $p_k(t)$ des Bilanzkreises an eben diesen verkaufen. Ein Angebotspreis unterhalb von $p_k(t)$ wäre ebenso sinnlos, da die Erzeuger ihren Strom dann besser direkt an den Bilanzkreis zu $p_k(t)$ verkaufen könnten. Daraus folgt:

$$p_a(t) \geq p_e(t) \geq p_k(t) \geq 0$$

Das Verhältnis von Preisen des Verbundnetzes und der Erzeuger ist unerheblich, da das Verbundnetz nie direkt, sondern immer nur indirekt über den Bilanzkreis in Erscheinung tritt.

Hier ein einfaches Beispiel: $p_a(t) = 2\,€/\text{kWh}$, $p_k(t) = 1\,€/\text{kWh}$

1. $p_e(t) > 2\,€/\text{kWh}$: Die Käufer würden günstiger beim BK kaufen. Der Anbieter könnte also nicht verkaufen.

2. $p_e(t) < 1\,€/\text{kWh}$: Der Anbieter könnte zu $p_k(t) = 1\,€/\text{kWh}$ an den BK besser verkaufen.

3. $2\,€/\text{kWh} \geq p_e(t) \geq 1\,€/\text{kWh}$: Bei diesem Preisintervall realisieren sowohl der Anbieter als auch der Käufer einen finanziellen Vorteil.

8.4. Vertragsabschluss

Dem Blackboard-Agenten werden von den Erzeuger-Agenten Angebote geschickt, die den Verbraucher-Agenten zugänglich gemacht werden. Ein Angebot besteht aus der *angebotenen Leistung*, dem *Preis*, dem *Zeitraum der Lieferung* und die *Dauer der Gültigkeit* des Angebotes. Die Angebote werden genau dann gelöscht, wenn der Gültigkeitszeitraum abläuft oder das Angebot vom Erzeuger zurückgezogen wird. Im zweiten Fall unterscheidet man das Entfernen bei Abschluss des Vertrags oder bei Anpassen des Angebots wegen einer Prognoseänderung beim Erzeuger. Dabei geht man davon

aus, dass die Agenten fair handeln, und das Anpassen von Angeboten nur durch die Prognosenänderungen verursacht werden kann.

Die Verbraucher-Agenten fordern vom Blackboard Angebote an, die bestimmten Bedingungen (Preis, Zeitraum, Leistung) genügen. Das Blackboard schickt den Agenten Listen mit den entsprechenden Angeboten. Durch den Benachrichtigungsmechanismus informiert das Blackboard selbständig die Agenten über die neuen bzw. gelöschten Angebote. Dafür müssen sich die Agenten beim Blackboard in eine Benachrichtigungsliste mit einem Filter eintragen.

Aus den passenden Angeboten können die Verbraucher-Agenten ihren Vorstellungen entsprechend auswählen. Sie stellen Kontakt zum zuständigen Erzeuger-Agenten her und übermitteln ihr eigenes Angebot. Dieses kann sowohl im Lieferzeitraum als auch in der Leistung, jedoch nicht im Preis variieren. Falls Zeitraum und Leistung von denen im Angebot abweichen, darf der Erzeuger die Nachfrage ablehnen.

Der Preis ist nicht direkt verhandelbar. Allerdings kann ein Verbraucher-Agent versuchen, einen günstigeren Preis zu erreichen, indem er das Angebot des Erzeugers nicht annimmt, und bis kurz vor den Startzeitpunkt des Angebotes wartet, und darauf spekulieren, dass der Erzeuger sein Angebot bis dahin nicht verkaufen kann und seinen Preis deshalb reduzieren wird.

Nimmt ein Verbraucher das Angebot eines Erzeugers vollständig in Anspruch, d. h. weder Lieferzeitraum noch gewünschte Leistung weichen vom Angebot des Erzeugers ab, so darf der Erzeuger den Vertrag nur dann ablehnen, wenn bereits ein anderer Verbraucher den Zuschlag erhalten hat.

Lehnt der Erzeuger ein Angebot eines Verbrauchers ab, kommt kein Vertrag zustande, der Verbraucher kann allerdings zu einem späteren Zeitpunkt ein erneutes Angebot abgeben. Wird ein Vertrag über nur einen Teil des ursprünglichen Angebots des Erzeugers abgeschlossen, so löscht der Erzeuger-Agent das alte Angebot und stellt entsprechend ein oder mehrere neue Angebote über den verbleibenden Rest an das Blackboard.

8.5. Verbundnetz und Bilanzkreisverantwortlicher

Das Verbundnetz (VN) sichert die Stromversorgung im Bilanzkreis dadurch, dass der Bilanzkreisverantwortlicher jederzeit zu einem Zukaufspreis $p_z(t)$ Leistung aus dem VN zukaufen kann, um sein eigenes Netz auszubalancieren. Ebenso besteht für den Bilanzkreisverantwortlichen bei einem Leistungsüberschuss die Möglichkeit, überschüssige Leistung an das VN zum Verkaufspreis $p_v(t)$ zu verkaufen. Beide Preise sind extern gegeben, ändern sich aber über den Tagesverlauf. Sie sind als Funktion in Abhängigkeit von der Zeit gegeben. Sie bestimmen das Preisniveau, mit dem der Bilanzkreis Strom anbietet bzw. kauft (siehe Abschnitt 8.3).

Eine weitere Aufgabe des Bilanzkreisverantwortlichen ist die Abrechnung der erfüllten Verträge zwischen Erzeugern und Verbrauchern genauso wie Informationsunterstützung von Erzeugern und Verbrauchern bezüglich der Zukaufs- bzw. Verkaufspreisen und bereits eingespeisten bzw. bezogenen Leistungsmengen vom VN. Ebenso sind – bei Abweichungen und nicht ausreichendem Angebot oder Nachfrage – Zukäufe und Verkäufe vom bzw. in das Verbundnetz abzurechnen. Hierzu müssen die jeweiligen Preise, die vereinbarten und die tatsächlichen Leistungsflüsse bekannt sein. Folgende Fälle sind möglich:

1. *Vertraglich vereinbarte Leistung P_{soll} und tatsächliche gelieferte P_{ist} stimmen überein:* Verbraucher zahlt vereinbarten Preis $p \cdot P$ an Erzeuger.

2. *Erzeuger konnte nur weniger erzeugen als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Erzeuger zahlt an den Bilanzkreis für zugekaufte Leistung $p_a \cdot (P_{\text{soll}} - P_{\text{ist}})$. Alternativ kann der Erzeuger die zu wenig erzeugte Leistung bei anderen Agenten zukaufen.

3. *Erzeuger produziert mehr Leistung als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Erzeuger erhält für zusätzlich eingespeiste Leistung $p_k \cdot (P_{\text{ist}} - P_{\text{soll}})$ vom Bilanzkreis.

4. *Verbraucher benötigt mehr Leistung als vereinbart:* Verbraucher zahlt

an Erzeuger den vereinbarten Preis $p \cdot P$, Verbraucher zahlt für zusätzliche Leistung an den Bilanzkreis $p_a \cdot (P_{\text{ist}} - P_{\text{soll}})$.

5. *Verbraucher benötigt weniger Leistung als vereinbart:* Verbraucher zahlt an Erzeuger den vereinbarten Preis $p \cdot P$, Verbraucher erhält für quasi eingespeiste Leistung $p_k \cdot (P_{\text{soll}} - P_{\text{ist}})$. Alternativ kann der Verbraucher die Leistung, die er nicht benötigt, weiterverkaufen.

6. *Verbaucher bezieht Strom ohne Vertrag:* entspricht dem Fall 4 mit $P_{\text{soll}} = 0$.

7. *Erzeuger produziert Strom ohne Vertrag:* entspricht dem Fall 3 mit $P_{\text{soll}} = 0$.

Die genannten Funktionen (Abrechnung, Informationbereitstellung und Repräsentanz des Verbundnetzes) können in einem Agenten realisiert werden.

8.6. Blackboard

Das Blackboard (BB) stellt eine elektronische Energiebörse dar, die den Erzeugern und Verbrauchern die Kommunikation für das Verhandeln ermöglicht, und gleichzeitig für einen ordentlichen Verlauf vom Einstellen bzw. Abfragen von Angeboten sorgt. Dies geschieht in Form eines Angebotshandels, wobei Agenten Energie verkaufen können und Angebote einstellen. Diese Angebote können von anderen Agenten gekauft werden.

Diese Funktionalität soll in einem Agenten realisiert werden.

8.7. Technik

Die Technik hätte in der realen Welt die Aufgabe einer Schnittstelle zwischen der Steuerungstechnik der Erzeuger/Verbraucher und dem Verhandlungsteil. So steuert die Technik den realen Verbrauch der Verbraucheragenten und die reale Produktionsmenge bei den Erzeuger-Agenten. Zudem stellt sie reale Wetterdaten zur Verfügung, welche aus der Datenbank geliefert werden. Darüber hinaus werden aus der Datenbank vorgegebene Wetterprognosen lokal von der Technik korrigiert.

8.8. Graphische Benutzeroberfläche

Ein wesentlicher Vorteil der zweiten Version gegenüber der ersten soll die einfachere Bedienung und Auswertung mittels einer graphischen Benutzeroberfläche sein.

Konfiguration: Die Konfiguration der Agenten kann nun über die graphische Oberfläche erfolgen, d.h., für die Verbraucher können die Kurven des Strombedarfs eingegeben werden, und für die Erzeuger die Prognosekurven, nach denen sie ihr Angebot berechnen.

Bedienung: Die Bedienung von DEZENTEN soll komplett über die graphische Oberfläche erfolgen können. Im einzelnen können die verschiedenen Entitäten des Systems über die GUI gestartet werden. Die Gesamtsimulation soll über die GUI sowohl gestartet als auch beendet werden können.

Auswertung: Die Auswertung soll komplett über die GUI erfolgen können. Die zuvor in eine Datenbank geschriebenen Daten der Simulation sollen übersichtlich als Graphen ausgegeben werden.

8.9. E/A-Schnittstellen

Alle Ein- bzw. Ausgabedaten für die Simulation werden zentral in einer Datenbank in Form von "Testkonfigurationen" abgelegt, wodurch das Eintragen der Simulationsparameter und das Auswerten der Simulation flexibler und zugänglicher gemacht werden.

8.9.1. Eingabedaten

Die Eingabedaten für unsere Simulation sind die Anzahl und der Typ der Agenten (sprich Bilanzkreisverantwortlicher, Blackboard, Erzeuger- und Verbraucher-Agenten), ihr Strategietyp, Verbrauchsprofile, Wetterprofile und die Zukaufspreis- bzw. Verkaufpreise für den Strom des Verbundnetzes (VN). Die Erzeugeragenten unterteilen sich in Brennstoffzellen, Solaranlagen und Windkraftanlagen. Jeder Agent hat einen vorgegeben Strategietyp, der seine Preisbildung während der Verhandlung bestimmt.

Die Verbrauchsprofile, Wetterprofile und die Zukaufs- bzw. Verkaufspreise werden in Form von Polynomkurven unterschiedlichen Grades angegeben. Für die Verbrauchsprofile gibt es sowohl den prognostizierten Verbrauch als auch den realen Verbrauch. Die Wetterprofile sind unterteilt in zwei Kategorien: Profile für Solaranlagen und Profile für Windkraftanlagen. Diese werden ebenfalls zweifach gegeben sowohl für die Prognose als auch für den realen Wetterverlauf. Die mögliche abzugebende Leistung der Brennstoffzellen wird durch die vorgegebene Leistung `maxPower` definiert. Die Zukaufs- und Verkaufspreise vom VN werden jeweils als prognostizierte und reale Werte angegeben.

8.9.2. Ausgabedaten

Zur Analyse der Simulation werden die für die Auswertung benötigten Daten von den Agenten in die Datenbank eingetragen, die anschließend durch die GUI in Form von Diagrammen veranschaulicht werden. Dabei werden alle Diagramme für einen beliebigen Simulationstag mit dem Minutenraster skaliert.

Um die ganze Interaktion zwischen unserem simulierten Netzsegment und dem VN nachvollziehen zu können, werden die gesamten vom VN zugekauften bzw. an das VN verkauften Leistungsmengen dargestellt. Zusätzlich wird die Differenz zwischen diesen beiden Mengen ausgegeben, als Maßstab dafür, wie gut das System ausbalanciert ist.

Zu jedem Verbraucher/Erzeuger werden je zwei Graphen erzeugt, die die eingekauften und verbrauchten bzw. verkauften und produzierten Leistungsmengen beschreiben. Diese Graphen werden in einem gemeinsamen Koordinatensystem angezeigt. Sie zeigen, wie gut der Agent seine Verhandlungsstrategien realisiert hat, d.h. deren Gewinn oder Verlust. Um die ganze Verhandlung beurteilen zu können, werden die gleichen Diagramme für die aufsummierten Leistungsmengen aller Agenten erstellt.

Die Agenten führen während der Simulation die Anpassung von Wetterprognosen an den realen Verlauf durch. Um ihre Anpassungfähigleit einzeln analysieren zu können, werden die ursprunglichen und korrigierten Wetterprognosen neben dem tatsächlichen Wetterverlauf dargestellt.

9. Pflichtenheft

Die in Abschnitt 8 genannten Anforderungen sind zu erfüllen. Alle Module sollen mittels TCP/IP kommunizieren, unabhängig davon, ob die Simulation auf einem oder mehreren Rechnern abläuft.

Zudem sollen alle Module ihre wichtigen Aktionen protokollieren und in der Datenbank abspeichern. Dies betrifft insbesondere das Erzeuger-Modul und das dazugehörige Technik-Modul, das Verbraucher-Modul und das Bilanzkreis-Modul. Daten sollen dabei auch redundant gesammelt werden, d. h. auch wenn z. B. Daten des Bilanzkreis-Moduls aus den beiden anderen bestimmt werden könnten, soll das Modul seine Daten dennoch protokollieren.

Darüber hinaus sind Testdaten für die Simulation sowie zu erwartende Ergebnisse festzulegen. Dies gilt besonders für die Preissituation, d. h. die Preise des Bilanzkreises einerseits sowie der Preise, zu denen Verbraucher letztendlich ihren Strom beziehen.

Die Testläufe sind dabei in drei größere Szenarien unterteilt. Im ersten sind grundlegende Funktionen der Simulation zu testen, im zweiten werden Prognosekorrektur und Nachverhandlung mit einbezogen und im dritten Testszenario ist zu testen, welche Auswirkungen die verschiedenen Agentenstrategien auf den Verlauf der Agenten haben. Es sind vier Strategien implementiert:

- *dangerous* Die Strategie *dangerous* ist eine Strategie, bei der ein erhöhtes Verlustrisiko eingegangen wird, wodurch aber auch die Möglichkeit, höhere Gewinne zu erzielen, gegeben ist.
- *mixture* Diese Strategie bietet mit mittlerem Risiko an.
- *no risk* Hier werden Angebote mit minimalem Risiko erstellt, d. h. dass der Preis für ein Angebot so berechnet wird, dass der Produzent keinen Verlust erleidet.

- *medium* Die Strategie *medium* berechnet einen Angebotspreis so, dass der Erwartungswert des Gewinns, den der Produzent durch sein Handeln erreicht, maximiert wird.

Es müssen sinnvolle Verhandlungen stattfinden, es darf also nicht die Situation eintreten, dass alle Verbraucher ihren Strom vom Bilanzkreis beziehen.

Die Auswertung der Simulation soll während der Simulation erfolgen können. Während der Simulation werden die verschiedenen Auswertungskurven automatisch generiert und können bei Bedarf angezeigt werden. Die Auswertung läuft zeitlich versetzt, d. h., dass die Simulationszeit dem Ende der Auswertungsperiode immer voraus ist.

10. Design und Implementation

10.1. Programmiersprache und Entwicklungsumgebung

Zu Beginn des ersten Semesters entschied sich die Gruppe für Java 1.4.2 (Sun Java 2 SE) für Linux als Implementationssprache. Da die Weiterentwicklung des Systems im zweiten Semester auf Basis der Implementation der ersten Phase erfolgte und der bestehende Code zum Teil weiterverwendet werden konnte, wurde die Implementationssprache beibehalten. Auch die mittlerweile vertraute Entwicklungsumgebung Eclipse der ersten Phase wurde weiterbenutzt.

10.2. Paket-Struktur

In der zweiten Phase gibt es elf Pakete, die in den folgenden Abschnitten beschrieben werden:

- Agent – Abschnitt 10.3
- BB – Abschnitt 10.4
- BRE – Abschnitt 10.5
- DezentenLib – Abschnitt 10.6
- DezentenLib.DataSource – Abschnitt 10.7
- DezentenLib.Event – Abschnitt 10.8
- DezentenLib.Net – Abschnitt 10.9
- Simulation – Abschnitt 10.10

- Simulation.DbWrapper – Abschnitt 10.12
- Simulation.GUI – Abschnitt 10.13
- Technics – Abschnitt 10.14

Die gesamte Klassenstruktur ist in Anhang B dargestellt.

10.3. Paket `Agent`

Das Paket `Agent` enthält die Klasse `Agent`, die zur Durchführung der Verhandlungen des Agenten dient.

10.3.1. Klasse `Agent`

Zur Funktionsweise des Startvorgangs und der Terminierung siehe Klasse `AbstractSimulationMember`, von der diese Klasse abgeleitet ist, im Abschnitt 10.6.1 auf Seite 93.

Im folgenden die Beschreibung einiger Methoden, die die Agenten brauchen, um Angebote und Nachfragen zu generieren, zu verhandeln und abzurechnen.

Konstruktor `Agent`

Der Konstruktor des `Agent` liest die Konfigurationsdatei ein. Der Agent wird hierbei als Verbraucher oder Erzeuger bestimmt. Im Falle eines Erzeugeragenten wird die Technik des Erzeugeragenten, siehe Abschnitt 10.14.3, als Brennstoffzelle, Solarzelle oder Windkraft initialisiert.

Methode `onStartSimulation`

In der Methode `onStartSimulation` wird die Strategie des Agenten festgelegt. Dies betrifft sowohl die eigendliche Angebots- und Verhandslungsstrategie als auch die Strategie der Wetterprognose. Es gibt insgesamt fünf Strategien, die dem Agenten zugewiesen werden können. Jede dieser Strategien setzt fünf grundlegende Parameter:

powerDrift

offerInterval Zeitraum (in Minuten), für den Angebote in der Zukunft erstellt werden.

exponent Exponent der Preisanpassungsfunktion in der Methode *modifyPrice*.

expirationfactor Teiler zur Bestimmung der Expiration des Offers (1<x<5)

intervalLength Maximale Länge von Angeboten.

Die genaue Zuordnung der Parameter ist Tabelle 10.1 zu entnehmen.

Strategie	powerDrift	offerInterval [Tage]	exponent	expiration-factor	interval-length [Min]	Prognosestrategie[a]
medium	0.10	1	1.0	2	60	gl. Durch. – optimal
no risk	0.20	1	0.5	2	240	gl. Durch. – no_risk
dangerous	0.05	1	4.0	4	30	gl. Durch. – dangerous
linear short	0.10	1	1.0	2	60	linear(10)
linear long	0.10	1	1.0	2	60	linear(20)

Tabelle 10.1.: Strategien der Agenten

[a] Für eine genaue Beschreibung der Prognosekorrekturen „gleitender Durchschnitt“ und „lineare Prognose“ und der zugehörigen Parameter *medium, no risk, dangerous* bzw. *zeitraum* siehe Abschnitt 10.14.6 und 10.14.5.

Hier nun eine genaue Beschreibung der Strategien und ihrer Wirkungsweise.

dangerous Die Strategie *dangerous* ist eine riskante Angebotserstellungsstrategie. Hier wird nur eine Abweichung von 5% nach unten von der prognostizierten Menge und dem kalkulierten Preis gestattet, was durch die Variable powerdrift festgesetzt wird. Die Preisanpassung erfolgt hier so, dass der Preis nach der Angebotserstellung zunächst nur geringfügig sinkt und erst kurz vor dem Lieferzeitpunkt des Angebots stark reduziert wird, wofür der Ausdruck exponent = 4 verantwortlich ist.

mixture Die Strategie *mixture* ist eine Strategie, die mit mittlerem Risiko anbietet und von den Einstellungsparametern zwischen *dangerous* und *medium* liegt. Falls keine Strategie gewählt wird, wird die Strategie auf *mixture* gesetzt.

no risk Die Strategie *no risk* ist eine Strategie, bei der die Wahrscheinlichkeit, dass der Producer Verlust erleidet, minimiert wird. Hervorzuheben ist, dass die Mengen und Preise eines Angebotes mit einem Algorithmus in der Form berechnet werden, dass sie in vielen Fällen deutlich unter den prognostizierten Werten liegen, um somit die Wahrscheinlichkeit eines nötigen Nachkaufes zu minimieren. Durch den Ausdruck powerdrift = 0.2 wird die Abweichung auf bis 20% nach unten begrenzt. Außerdem verläuft die zeitliche Preisanpassung der Angebote derart, dass der Preis schon früh relativ stark gesenkt wird, um die Angebote wenigstens mit etwas Gewinn verkaufen zu können. Durch `exponent` = 0.5 wird diese Eigenschaft sichergestellt.

medium Bei der Strategie *medium* wird die Kombination der Parameter so gesetzt, das der Erwartungswert des Gewinns, den der Producer durch sein Handeln erreicht, maximiert wird. Das wird dadurch erreicht, dass die zeitliche Preisanpassung linear erfolgt und die Mengen und Preise eines Angebotes mit einem Algorithmus in der Form berechnet werden, dass sie immer knapp (maximal 10%) unter den prognostizierten Werten liegen, entsprechend den Werten `exponent` = 1 und powerdrift = 0.1.

Methode `makeBuyOffer`

Die Methode `makeBuyOffer` dient zum Erzeugen von Kaufangeboten und wird von der Technik aufgerufen. Es werden die Netzadresse, die ID des anfragenden Clients und das Offer als Parameter übergeben. Es wird zunächst überprüft, ob das ausgewählte Angebot gültig ist.

Methode `removeOffer`

Die Methode `removeOffer` löscht das durch die OfferID eindeutig bestimmte Offer vom Blackboard. Hierbei wird auch der Grund des Löschens als Parameter gesetzt; die möglichen Gründe sind Löschen wegen Verkauf, Löschen wegen Prognoseänderung oder Löschen, weil der Erzeuger den Preis seines Angebots modifizieren möchte.

Methode `initIntervalTables`

Diese Methode dient zur Initialisierung der durch den `Agent` erzeugten, leeren Tabellen.

Methode `setTimeOfInterval`

In dieser Methode wird aus der Menge der Angebote der frühste Zeitpunkt bestimmt, zu dem Angebote existieren, die das komplette Zeitintervall, für das am Blackboard angefragt wurde, abdecken. Ist kein solches Angebot vorhanden, wird der Zeitpunkt gewählt, der eine größtmögliche Deckung mit den Angeboten darstellt.

Methode `calcBidFromOfferlist`

Die Methode `calcBidFromOfferlist` wählt die Angebote aus, zu denen ein Kaufangebot geschickt werden soll. Dabei vergleicht sie die Zeitintervalle, für die Strom gekauft werden muss mit den Zeitintervallen der Angebote der Offerlist. Das Angebot, das die größtmögliche zeitliche Deckung zwischen Bedarf und Angebot eines Intervalls bietet, wird zuerst genommen. Das Angebot mit der zweitgrößten Deckung als nächstes, usw.

Methode `chooseOffersToBuy`

In dieser Methode werden die Angebote ausgewählt und an die BidList übergeben, die für die Zeiträume, für die Strom gekauft werden soll, am günstigsten sind. Es werden genau soviele Angebote gewählt, bis der gesamte Energiebedarf gedeckt ist. Alle weiteren Angebote werden aus der lokalen temporären Offerlist gelöscht.

Methode `acceptBid`

Die Methode `acceptBid` überprüft, ob eine Nachfrage angenommen werden kann. Im positiven Fall wird intern eine Bestätigung erzeugt, im negativen Fall eine Absage.

Methode `generateOffer`

Die Methode `generateOffer` dient zur Erzeugung von Angeboten aus der Prognose. Hierbei wird für ein zuvor eingestelltes Zeitintervall (z.B. sechs Stunden) die Prognose vom Technikmodul geholt und gespeichert. Nun wird ein Schwankungsintervallwert festgelegt. Das ist ein Wert, der angibt, wie groß die prozentuale Abweichung der Leistung maximal sein kann, damit die Zeitpunkte zu einem Intervall zusammengefasst werden. Mit dieser Vorgabe werden die Werte der Prognose miteinander verglichen und die entstehenden Zeitintervalle zu Angeboten zusammengefasst. Diese Methode wird intern aufgerufen.

Methode `calculateOfferPrice`

Diese Methode berechnet den ersten Preis eines Angebotes anhand des prognostizierten BRE-Verkaufspreises zum Startzeitpunkt des Angebotes.

Methode `modifyPrice`

Die Methode steuert die Preise als Funktion der Zeit. Der Preis nähert sich, je näher der Startzeitpunkt des Angebotes kommt, asymptotisch an den BRE-Ankaufspreis an, ohne ihn dabei zu erreichen.

Methode `compareOffer`

Die Methode `compareOffer` vergleicht das übergebene Angebot mit eigenen Angeboten am Blackboard um evtl. Angebote zusammenzufassen. Danach werden die resultierenden Offer zum Blackboard gesendet. Falls im jeweiligen Intervall noch keine eigenen Offer am Blackboard stehen, wird das neue Offer zum Blackboard geschickt. Ansonsten wird das neue Offer mit den vorhandenen verknüpft. Diese Verknüpfung kann sowohl mengenmäßig erfolgen, das heißt, dass bei zwei zeitgleichen Angeboten die Leistungen addiert werden, als auch zeitmäßig, das heißt, das zwei aufeinanderfolgende Angebote aneinandergefügt werden.

Methode `modifyDifferenceTable`

Die Methode `modifyDifferenceTable` ändert für den Zeitraum, in dem ein Angebot erstellt wird, die Werte in der Tabelle, welche angibt, wieviel Leistung in jeder Minute des Tages verfügbar ist beziehungsweise benötigt wird.

Methode `getExpirationtime`

Die Methode `getExpirationtime` ermittelt die Gültigkeitsdauer eines Angebotes. Die jeweilige Gültigkeitsdauer der Angebote sinkt, je näher man sich an den Startzeitpunkt des Angebots nähert. Zum Beispiel wird ein Angebot, das vier Stunden gültig war und jetzt abgelaufen ist, anschließend mit einer Gültigkeitsdauer von zwei Stunden neu erstellt.

Methode `getOffers`

Diese Methode realisiert die Suche nach neuen Angeboten. Zu jeder Suche wird ein Notify beim Blackboard registriert (siehe Abschnitt 10.4.1). Alte bzw. nicht mehr benötigte Angebote werden aus der internen Liste gelöscht.

Methode `neededEnergy`

Diese Methode berechnet eine Tabelle, die die noch benötigte Energie angibt. Es wird also ein Wert berechnet, der sich ergibt als ursprünglich benötigte Energie abzüglich der gekauften und der angefragten Energie.

Methode `isEnergyNeeded`

Diese Methode prüft, ob für ein übergebenes Interval noch Energie gekauft werden soll.

Methode `generateSearchTable`

Die Methode generiert eine Tabelle, welche angibt, für welche Minuten heute und morgen Energie gekauft werden muss.

Methode `modifyOfferData`

Diese Methode sorgt für die Aktualisierung der Angebote des Erzeugers am Blackboard, nachdem eine Nachfrage des Verbrauchers angenommen wird. Die verkaufte Menge wird von der angebotenen Menge abgezogen, wobei mehrere Fälle unterschieden werden. Aus einem Offer können in dieser Methode bis zu drei neue entstehen: Wenn zum Beispiel aus einem dreistündigen Angebot in der mittleren Stunde die Hälfte der angebotenen Leistung herausgekauft wird, so wird das alte Angebot von Blackboard gelöscht und es werden drei neue Angebote eingestellt: Für die erste und letzte Stunde je ein Angebot mit der ursprünglich Angeboten Menge, für die mittlere Stunde ein Angebot mit der Hälfte der Leistung.

Methode `putOffer`

Die Methode `putOffer` sendet ein Angebot zum Blackboard.

Methode `getBRECurrentBuyPrice`

Die Methode `getBRECurrentBuyPrice` liefert den aktuellen Ankaufspreis für eine kWh Strom des `BRE`.

Methode `validateOffer`

Diese Methode wird von `makeBuyOffer` aufgerufen und bekommt ein Offer-Objekt als Parameter. Die Methode prüft nun, ob im durch das Offer-Objekt beschriebenen Nachfrageintervall die gewünschte Leistung noch verfügbar ist.

Methode `getBRESellPricesByDate`

Die Methode `getBRESellPricesByDate` holt für einen als Parameter bestimmten Tag die Prognose der Preise, zu denen der Bilanzkreis Strom verkauft. Die insgesamt 1440 Werte für den bestimmten Tag werden anschließend in einer Tabelle gespeichert.

Methode `getBREBuyPricesByDate`

Die Methode `getBREBuyPricesByDate` holt für einen als Parameter bestimmten Tag die Prognose der Preise, zu denen der Bilanzkreis Strom kauft. Die insgesamt 1440 Werte für den bestimmten Tag werden anschließend in einer Tabelle gespeichert.

Methode `getConsumerPrognosis`

Diese Methode holt die Verbraucher-Prognose, in der der Bedarf prognostiziert ist, von der Technik.

Methode `getProducerPrognosis`

Diese Methode holt die Erzeuger-Prognose, in der die zu erwartende erzeugte Energie prognistiziert ist, von der Technik. Die prognostizierten Werte sind im Falle einer Solar- oder Windkraftanlage als die Wetterprognose, die mit größter Wahrscheinlichkeit eintreffen wird, zu verstehen.

Methode `buyNSell`

Die Methode `buyNSell` wird periodisch von `AgentThread` aufgerufen. Sie holt die Prognose vom Technikmodul und überprüft, ob Abweichungen von der vorherigen Prognose festgestellt wurden. Außerdem wird die Methode `checkExpiration` aufgerufen, um die Angebotsliste nach abgelaufenen Angeboten zu durchsuchen. Wenn Handlungsbedarf (entweder Prognoseänderung oder abgelaufene Angebote am Blackboard) festgestellt wurde, wird die Methode `reactToChanges` aufgerufen.

Methode buy

Die Methode versucht, für alle übergebenen Angebote einen Kaufvertrag einzugehen.

Methode checkExpiration

Diese Methode prüft das Ablaufsdatum jedes angebotenen Angebotes und ruft, falls dieses erreicht ist, eine neue Preiskalkulation auf. Dies bedeutet, dass dieses Angebot vom Blackboard gelöscht wurde und gegebenenfalls mit neuem Preis wiedereingestellt wird.

Methode updateOffer

Diese Methode wird von `checkExpiration` aufgerufen und aktualisiert den Preis des Angebotes. Danach wird das Angebot erneut am Blackboard eingestellt.

Methode doAccounting

Die Methode `doAccounting` ruft die Abrechnung mit dem BRE im Technik-Modul auf.

10.4. Paket BB

Das Paket `BB` enthält sowohl die Klasse `BlackBoard` als auch das Interface `BlackboardDispatcherTarget` und die Hilfsklassen `AgentWrapper`, `NotifyAgentsThread` und `Statistics`.

Das Interface `BlackboardDispatcherTarget` enthält alle Schnittstellemethoden, die das Blackboard den Agenten zur Verfügung stellt.

10.4.1. Klasse BlackBoard

Zur Funktionsweise des Startvorgangs und der Terminierung siehe Klasse `AbstractSimulationMember`, von der diese Klasse abgeleitet ist, im Abschnitt 10.6.1 auf Seite 93.

Die Klasse `BlackBoard` verwaltet alle Angebote der Agenten. Diese können sowohl neue Angebote einstellen als auch alte Angebote löschen (z.B.

nach Vertragsabschluß oder aufgrund einer Prognoseänderung). Angebote, deren Gültigkeit abgelaufen ist, werden automatisch vom Blackboard entfernt.

Zudem verwaltet die Klasse eine Benachrichtigungsliste. Alle Agenten, sowohl Erzeuger als auch Verbraucher, können sich in diese Liste eintragen und werden durch Setzen eines Filters nur über die sie interessierenden Angebote automatisch durch eine Notifynachricht informiert. Die Notify-Nachrichten werden direkt nach dem Einstellen bzw. Löschen des Angebots mittels der `NotifyAgentsThread` Klasse abgesendet. Gefiltert werden kann nach dem Gültigkeitszeitraum von Angeboten, nach Preis und Leistung.

Über die Klasse `Statistics` verwaltet das Blackboard die Statistiken über die Angebote. Zum einen können die Mengen von verkaufter Energie, zum anderen die Preise (niedrigste, höchste, mittelwertige) über einen bestimmten Tag abgefragt werden.

Die für die Auswertung der Simulation wesentlichen Ereignisse am Blackboard werden anhand der entsprechenden Methoden der `BBEvent` Klasse in die Auswertungsdatenbank eingetragen.

Eingehende Nachrichten werden vom `BlackboardReceiver` empfangen, siehe Abschnitt 10.9.4, der die folgenden Methoden aufruft.

Konstruktor `BlackBoard`

Der Konstruktor der Klasse `BlackBoard` instantiiert seinen `Blackboard-Receiver` und erzeugt zuerst die leeren Listen, in die später die Angebote, Notifyfilter, abgeschlossene Verträge und Statistiktabellen eingetragen werden.

Methode `getOffers`

Anhand des übergebenen Filters liefert die Methode eine Liste passender Angebote zurück, so dass Agenten sich auch jederzeit aktiv informieren können und nicht allein auf den Benachrichtigungsmechanismus angewiesen sind.

Methode `putOffer`

Die Methode `putOffer` wird von Erzeuger-Agenten benutzt, um neue Angebote am Blackboard zu veröffentlichen. Dabei wird die Gültigkeit des An-

gebots geprüft, ob das Angebot mit der gleichen `ID` bereits existiert und ob Endzeit und Ablaufzeit gültige Werte haben. Anschließend werden alle an diesem Angebot interessierten Agenten anhand der `NotifyAgentsThread` über das neu eingestellte Angebot informiert.

Methode `delOffer`

Die Methode löscht das Angebot mit der übergebenen `ID` aus der Angebotsliste. Falls das Angebot zu einem Vertrag abgeschlossen wurde, wird es in der Liste der abgeschlossenen Verträge gespeichert und beeinflusst die Statistik über die verkauften Leistungsmengen. Anschließend werden alle in der Benachrichtigungsliste stehenden Agenten (beliebiger `Filter` gesetzt) anhand der `NotifyAgentsThread` über das gelöschte Angebot informiert.

Methode `setNotify`

Agenten, die automatisch über die Änderungen der Angebotsliste informiert werden möchten, können sich mit Hilfe dieser Methode für Benachrichtigungen anmelden, indem sie einen Filter setzen, für Angebote, die sie interessieren. Beim `Filter` können Start- und Endzeit, minimale bzw. maximale Preise und Leistung als Kriterien zum Filtern definiert werden. Es gibt immer nur einen Filter pro Agent, der mittels der `AgentWrapper` am Blackboard gespeichert ist. Bei Setzen eines Filters wird der alte Filter ersetzt.

Methode `delNotify`

Mit dieser Methode können sich Agenten vom Benachrichtigungsdienst wieder abmelden. Nach dem Aufruf der Methode wird der Agent weder über die neuen noch über die gelöschten Angebote benachrichtigt.

Methode `getStatisticOfSold`

Die Methode stellt die Statistik über die verkauften Leistungsmengen in Form einer Tabelle mit Minutenraster für einen vom Agenten angeforderten Tag zur Verfügung. Dabei enthält die Tabelle die aufsummierten Leistungsmengen zu jeder Minute und keine Information über die abgeschlossenen

Verträge wie z.B. Agenten oder Preise. Die Aktualisierung und Berechnung der Statistiken erfolgt über die Hilfsklasse `Statistics`.

Methode `getStatisticOfPrice`

Die Methode stellt die Statistik über die Angebotspreise in Form von drei Tabellen mit Minutenraster zur Verfügung. Die drei Tabellen stehen für den niedrigsten, höchsten und mittelwertigen (über die Menge) Preis zu jeder Minute des vom Agenten angeforderten Tages. Die Aktualisierung und Berechnung der Statistiken erfolgt über die Hilfsklasse `Statistics`.

10.5. Paket `BRE`

Das Paket `BRE` enthält sowohl das Interface `BREDispatcherTarget` als auch die Klasse `BalanceResponsibleEntity` für den BalanceResponsibleEntity-Agenten, der die Aufgaben des Bilanzkreisverantwortlichen übernimmt:

- Liefert den aktuellen Preis für die nächste Minute für den An-/Verkauf an das Verbundnetz über den Bilanzkreis.
- Liefert die Preistabelle für einen bestimmten Tag.
- Abrechnung der Verträge und Lieferungen eines Tages.
- Führung und Bereitstellung von Statistiken über die Einspeisung bzw. Bezug der Leistung des Verbundnetzes.

10.5.1. Klasse `BalanceResponsibleEntity`

Zur Funktionsweise des Startvorgangs und der Terminierung siehe Klasse `AbstractSimulationMember`, von der diese Klasse abgeleitet ist, im Abschnitt 10.6.1 auf Seite 93.

Die Klasse `BalanceResponsibleEntity` modelliert den Agenten, der die prognostizierten und realen Ein- und Verkaufspreise des Verbundnetzes verwaltet. Bei der Abfrage von Erzeuger- oder Verbraucher-Agenten gibt er Auskunft über die Preise (aktuelle oder prognostizierte) und führt die Abrechnung zwischen Soll- und Istwerten der abgegebenen bzw. bezogenen Leistung durch. Außerdem stellt er bei der Abfrage den Agenten die

Information über die ins Verbundnetz eingespeiste bzw. bezogene Gesamtleistung für ein angefordertes Zeitintervall zur Verfügung.

Die Preise des Verbundnetzes werden von der Klasse `DataSource` aus der Konfigurationsdatenbank online geliefert und intern verwaltet.

Alle für die Auswertung der Simulation wesentlichen Ereignisse vom BalanceResponsibleEntity-Agenten werden mittels der entsprechenden Methoden der `BREEvent` Klasse in die Auswertungsdatenbank eingetragen.

Für die Reaktion von eingehenden Nachrichten ist der Thread der Klasse `BREDispatcherTarget` zuständig.

Konstruktor `BalanceResponsibleEntity`

Der Konstruktor der Klasse `BalanceDistrict` instantiiert seinen `BREReceiver` und erzeugt die Datenstrukturen für das Protokollieren von Leistungsmengen.

Methode `getPrice`

Über diese Methode kann der Ein- und Verkaufspreis des Verbundnetzes für die nächste Minute der Simulation ermittelt werden. ähnliche Methoden `getBRESellPrice` und `getBREBuyPrice` liefern nur den Ver- bzw. Einkaufspreis für die nächste Minute.

Methode `getPriceTable`

Die Methode `getPriceTable` liefert die Preistabelle (mit 1440 Einträgen) für den gegebenen Tag zurück. Falls das übergebene Datum der aktuelle oder ein zukünftiger Tag ist, werden die prognostizierten Preise zurückgegeben, andernfalls liefert die Methode die realen Preise. über den Parameter `priceTyp` wird bestimmt, ob es sich um Ein- oder Verkaufspreise handelt.

Methode `getBilling`

Alle Verbraucher- und Erzeuger-Agenten benutzen diese Methode, um eingespeiste oder bezogene Leistung abzurechnen. Dabei stehen positive Werte für ein Guthaben und negative Werte für an den Bilanzkreis zu zahlende Beträge.

Methode `getSoldPower`

Methode `getBoughtPower`

Diese Methoden liefern die Leistungsmengen (in Form einer Tabelle mit Minutenraster), die im angeforderten Zeitraum in das Verbundnetz eingespeist bzw. bezogen wurden. Da die Leistungsmengen erst nach jeder Abrechnung aktualisiert werden, können diese Statistiken von den aktuellen Werten abweichen.

Methode `getDifferenceInBRE`

Diese Methode liefert die Differenzen (in Form einer Tabelle mit Minutenraster) zwischen den in das Verbundnetz eingespeisten und aus dem Verbundnetz bezogenen Leistungsmengen für den angeforderten Zeitraum. Dabei stehen positive Werte für einen Überschuß von eingespeister Energie innerhalb des Bilanzkreises bzw. negative für einen Energiebedarf. Da auch diese Leistungsmengen erst nach jeder Abrechnung aktualisiert werden, können diese Statistiken von den aktuellen Werten abweichen.

10.6. Paket `DezentenLib`

Das Paket `DezentenLib` enthält einfache Hilfsklassen, um wichtige Entitäten wie Angebote oder Datentabellen zu kapseln und zudem drei Unterpakete, nämlich `DezentenLib.DataSource`, `DezentenLib.Event` und `DezentenLib.Net`, die für den Umgang mit Splines, für die Kommunikation mit der Datenbank sowie für die Netzwerkkommunikation zuständig sind.

Die folgenden Klassen dienen ausschließlich der Modellierung wichtiger Entitäten und bestehen im Wesentlichen nur aus öffentlichen Attributen:

`AbstractSimulationMember` ist eine Basisklasse aller Simulationsteilnehmer, die die Prozedur zum Starten und Beenden der Simulation vereinheitlicht.

`AgentType` enthält Konstanten, um Agenten und ihre Strategien zu unterscheiden.

`ClientID` kapselt die (global eindeutige) Kennung eines Agenten.

DataSourceType enthält Konstanten, um Kurventypen zu unterscheiden.

Filter enthält alle Daten, die notwendig sind, um aus der Liste aller Angebote am Blackboard einige, für den Anfragenden interessante, auszuwählen.

ID kapselt die (global eindeutige) Kennung von Angeboten.

OfferEventType enthält Konstanten zur Unterscheidung von Ereignissen, die mit Angeboten zu tun haben.

Offer modelliert Angebote.

PowerEventType enthält Konstanten zur Unterscheidung von Ereignissen, die mit Energie zu tun haben.

WorkerThread ist eine Hilfsklasse, die einen Thread zur Verfügung stellt, der in regelmäßigen (konfigurierbaren) Intervallen Aktionen anstößt.

Darüber hinaus gibt es noch die Klassen `DDate`, `Table` und `OfferList`, die im folgenden ausführlicher beschrieben werden. Zusätzlich gibt es die drei Comparator-Klassen `OfferPriceComparator`, `OfferRankComparator` und `OfferTimeComparator`, die jeweils zwei Angebote anhand der entsprechenden Attribute vergleichen.

10.6.1. Klasse `AbstractSimulationMember`

Zur Vereinheitlichung des Startvorgangs bzw. der Terminierung der Prozesse der Simulationsteilnehmer `Agent`, `BlackBoard` und `BalanceResponsibleEntity` dient diese abstrakte Basisklasse. Sie stellt die jeweils benötigte `main`-Methode sowie zu überschreibende Methoden, nämlich `onStartSimulation` und `onTerminateSimulation`, zur Verfügung. Zusätzlich existiert die abstrakte Methode `work`, die mit Hilfe der Klasse `WorkerThread` in einstellbaren Zeitintervallen aufgerufen wird und dazu benutzt werden kann, wiederkehrende Aufgaben in den erbenden Klassen zu erledigen.

Das von der Klasse `AbstractSimulationMember` implementierte Interface `DispatcherTarget` dient dazu, auf Nachrichten des Controllers reagieren zu können, bei dem sie sich automatisch registriert. Die Methode

`startSimulation` wird bei Empfang der Start-Nachricht durch den Receiver aufgerufen und signalisiert dieses Ereignis den erbenden Klassen durch Aufruf der Methode `onStartSimulation`. Anschließend wird der `WorkerThread` gestartet.

Die Terminierung findet in zwei Stufen statt. Zuerst wird nach Empfang einer Shutdown-Nachricht die Methode `shutdown` aufgerufen und damit der `WorkerThread` beendet. Dieses Ereignis wird nicht an erbende Klassen weitergegeben. Abschließend wird mit Empfang der Terminate-Nachricht die Methode `terminate` aufgerufen, die wiederum dieses Ereignis mittels `onTerminateSimulation` bekannt gibt und den Prozess beendet.

Methode `main`

Zunächst werden die Kommandozeilenparameter, nämlich die Adresse des Controllers sowie der Typ des zu startenden Simulationsteilnehmers, ausgewertet. Anschließend wird ein Objekt der entsprechenden Klasse instanziiert. Die eigentliche Arbeit wird von nun an durch den `WorkerThread` bzw. die vom Receiver gestarteten Threads ausgeführt, so dass nach der Instanziierung des Simulationsteilnehmers hier nur noch auf das Ende der Simulation gewartet wird.

10.6.2. Klasse `DDate`

Die Klasse `DDate` hat die Aufgabe, alle Grundfunktionen der normalen Date-Klasse von Java zu implementieren und zusätzlich eine Funktionalität bereitzustellen, die es ermöglicht, die Zeit beschleunigt ablaufen zu lassen, so dass Simulationen schneller als in Realzeit durchführbar sind. Um dies zu erreichen, wird in `DDate` die Systemzeit immer mit Hilfe eines Zeitfaktors umgerechnet, der zu Beginn der Simulation gesetzt wird. Die Umrechungsfunktion `calculateTime` berechnet dann mit Hilfe dieses Zeitfaktors und des Startzeitpunktes der Simulation die simulierte Zeit.

Methode `calculateTime`

Diese Funktion führt die eigentliche Berechnung der simulierten Zeit durch. Für die Berechnung der simulierten Zeit benötigt diese Methode die Startzeit der Simulation, den Beschleunigungsfaktor und die reale Zeit. Die

Startzeit und die reale Zeit werden innerhalb dieser Methode als Millisekunden seit dem 1.1.1970 00:00:00 Uhr angegeben. Die Umrechnungsformel ist:

$$\begin{aligned}\text{Simulierte Zeit} = \quad & (\text{RealeZeit} - \text{SimulationsStartzeitpunkt}) * \text{Zeitfaktor} \\ & + \text{SimulationsStartzeitpunkt}\end{aligned}$$

10.6.3. Klasse `Table`

Diese Klasse wird häufig in der Technik gebraucht und ist eine Wrapperklasse für ein Array, das für die Darstellung von Energie oder Kennlinien benutzt wird. Sie liefert zusätzlich zu den eigentlichen Kennlinien, die in dem Array abgespeichert werden, auch Information darüber, was in der `Table` abgespeichert ist. Deshalb gibt es ein Attribut *typ*, in dem steht, um welche Art von Daten es sich handelt. Bei dem Typ „kWmin“ (Kilowattminute) oder „kWh“ (Kilowattstunde) ist es auch wichtig, den Beginn des Gültigkeitszeitraums mit anzugeben, da diese Typen meistens einen Energieverbrauch darstellen. Dieser Startzeitpunkt wird in einem `DDate`-Objekt gespeichert.

Der Energieverbrauch wird immer minutenweise notiert, so dass für jede Minute ein Wert in der `Table` bzw. dem Array steht. Zusätzlich zu dieser Funktionalität als Datenstruktur besitzt `Table` einige Funktionen, um die alltägliche Arbeit mit diesen Objekten zu erleichtern, z. B. `addTable` und `subTable`.

Methode `addTable`

Diese Methode addiert zu dem aufrufenden Objekt ein anderes `Table`-Objekt. Dabei ist zu beachten, dass hierbei die Addition so durchgeführt wird, dass nur die Werte addiert werden, die sich zeitlich überlappen. Falls sich die beiden Tabellen zeitlich nicht oder nur teilweise überlappen dann werden keine Werte bzw. nur Teile addiert.

Methode `subTable`

Diese Methode funktioniert nach dem gleichen Schema wie die Addition, nur dass sie keine Werte addiert, sondern subtrahiert.

10.6.4. Klasse `OfferList`

Die Klasse `OfferList` stellt eine dynamische Datenstruktur für den einhetlichen Umgang mit den Angeboten dar. Dies wird von den Agenten sowohl bei der Übergabe als auch bei der internen Bearbeitung der Angebote verwendet.

Methode `getByID`

Die Methode liefert das Angebot mit dem übergebenen ID zurück.

Methode `removeByID`

Die Methode löscht das Angebot mit dem übergebenen ID aus der Liste.

Methode `getAsTable`

Die Methode bildet aus der Liste von Angeboten eine `Table` mit *starttime*, *stoptime* und *power*.

Methode `removeUnneededOffers`

Diese Methode löscht die Angebote aus der Liste , die ausserhalb des übergebenen Intervals liegen.

10.7. Paket `DezentenLib.DataSource`

Zur Abstraktion von Kennlinien und Kurven im Allgemeinen dienen die folgenden Klassen aus dem Paket `DezentenLib.DataSource`:

`ConstDataSource`

`DataSource`

`IntervalDataSource`

`NoDataException`

`NoFutureException`

Point

Pointlist

PointXComparator

SimpleInterval

Spline

Kernelement ist die Klasse `Spline`, die sämtliche Kurven repräsentiert. Zum Zugriff auf die Daten dient das Interface `DataSource`, welches von der Klasse `ConstDataSource` implementiert wird. Die Daten zur Definition eines Splines werden aus einer Datenbank entnommen (siehe auch Abschnitt 10.11 auf Seite 107).

Die Klasse `Point` ist die Basis für Kurven und dient der Kapselung eines zweidimensionalen (Stütz-)Punktes. Die Stützpunkte einer Kurve werden in der Klasse `Pointlist` verwaltet und können mit Hilfe von `PointXComparator` nach ihrer x-Koordinate aufsteigend sortiert werden.

Beim Zugriff auf Daten einer Kurve kann die Ausnahme `NoDataException` auftreten, falls die angegebene x-Koordinate außerhalb des Definitionsbereichs der Kurve liegt oder nicht ausreichend viele Stützpunkte zur Definition der Kurve vorhanden sind. Die Ausnahme `NoFutureException` wird geworfen, falls ein Agent unberechtigterweise auf in der Zukunft liegende Daten zugreift, z. B. im Fall von realen Wetterdaten (im Gegensatz zu Wetterprognosen).

10.7.1. Klasse `ConstDataSource`

Die Klasse `ConstDataSource` kapselt jeweils ein Spline-Objekt und dient den Agenten als Schnittstelle zu den Kurvendaten. Zudem fordert sie die benötigten Informationen zur Konstruktion einer Spline vom Controller an, der sie aus der Datenbank liest (siehe auch Abschnitt 10.10 auf Seite 106).

Mit Hilfe der im Interface `DataSource` definierten Methoden `getExactValue` sowie `getRoundedValue` kann auf die Daten der Spline zugegriffen werden.

10.7.2. Klasse `IntervalDataSource`

Aufgrund übernommenen Codes aus der ersten Phase wurde diese Klasse geschrieben, um mehrere variable Intervalle für Verbraucher-Agenten in einer Datenquelle zu verwalten – dies ist aufgrund gewachsener interner Strukturen nicht mit der `ConstDataSource` möglich gewesen.

Es stehen Methoden zur Abfrage der Anzahl der Intervalle, der Start- und Endzeiten sowie der Intervalldaten zur Verfügung.

10.7.3. Klasse `Spline`

In der Klasse `Spline` ist eine Variante des de-Boor Algorithmus zur Berechnung von B-Splines implementiert. Der wesentliche Unterschied ist, dass die Stützpunkte nach x-Koordinaten aufsteigend sortiert werden und zudem die Knotenpunkte identisch mit den x-Koordinaten der Stützpunkte sind.

Unterstützt werden unterschiedliche Grade für die B-Splines, eine beliebige Anzahl von Stützpunkten sowie optional zyklische Splines.

10.8. Paket `DezentenLib.Event`

Das Paket `DezentenLib.Event` beinhaltet Klassen, die den Ablauf der Simulation zur späteren Auswertung in der Datenbank speichern. Die Klassen enthalten nur statische Methoden.

10.8.1. Klasse `Event`

Die Klasse `Event` ist die Basisklasse der anderen Eventklassen. Sie enthält Methoden, die die Datenbankverbindung herstellen sowie einige Parameter setzen. Außerdem enthält sie Methoden, um Meldungen auszugeben.

Methode `initialize`

Diese Methode stellt die Verbindung zur Datenbank her. Außerdem werden die Parameter *debugLevel* und *tracing* vom Controller bezogen. *debugLevel* gibt an, ab welcher Signifikanz Meldungen ausgegeben werden. Es stehen sieben Level zur Verfügung. *tracing* gibt an, ob die einzelnen Methoden

zusätzlich zu dem Eintrag in die Datenbank, den sie vornehmen, die ihnen übergebenen Parameter als Meldung ausgeben sollen, um den Verlauf der Simulation besser verfolgen zu können. Diese Methode muß vor allen anderen Methoden, mit Ausnahme von `debug`, aufgerufen werden.

Methode `debug`

Diese Methode dient zum Ausgeben von Meldungen. Die Meldung wird nur ausgegeben, wenn der übergebene *debugLevel* größer oder gleich ist als der aktuell eingestellte *debugLevel*.

Methode `startAgent`

Methode `stopAgent`

Diese beiden Methoden werden aufgerufen, wenn ein Agent bzw. das Blackboard oder der BKV gestartet werden bzw. beendet werden. Es wird dann eine entsprechende Meldung ausgegeben.

10.8.2. Klasse `AgentEvent`

Die Klasse `AgentEvent` enthält Methoden, die die von den Agenten generierten Ereignisse in der Datenbank speichern. Es werden Ereignisse bezüglich eines Angebotes gespeichert, der Bilanzwert des Agenten für einen Tag wird gespeichert, ebenso werden veränderte Prognosen und Ereignisse bezüglich der Erzeugung und des Verbrauchs bzw. des Ankaufs und Verkaufs von Energie gespeichert.

Methode `addOffer`

Diese Methode wird aufgerufen, wenn ein Angebot (Offer) an das Blackboard gesendet wird.

Methode `removeOffer`

Diese Methode wird aufgerufen, sobald ein Angebot vom Blackboard entfernt wird.

Methode `gotBidForOffer`

Diese Methode wird aufgerufen, sobald der Agent ein Kaufgebot für eines seiner Angebote erhalten hat.

Methode `acceptBid`

Diese Methode wird dann aufgerufen, wenn er auf das Kaufgebot eines anderen Agenten eingeht.

Methode `rejectBid`

Diese Methode signalisiert die Ablehnung eines Kaufangebotes.

Alle diese Methoden werden von einem Agenten aufgerufen, der etwas anbietet.

Methode `bidForOffer`

Diese Methode wird dann aufgerufen, wenn der Agent ein Gebot für eines der Angebote am Blackboard abgibt.

Methode `successfulBid`

Diese Methode wird aufgerufen, sobald ein Gebot für ein Angebot am Blackboard vom anbietenden Agenten akzeptiert wurde.

Methode `unsuccessfulBid`

Diese Methode signalisiert ein erfolgloses Gebot.

Alle diese Methoden werden von einem Agenten aufgerufen, der an einem Angebot eines anderen Agenten interessiert ist.

Methode `accountValueOfDay`

Mit dieser Methode wird der Gewinn bzw. Verlust eines Agenten für einen Tag gespeichert.

Methode changePrognosis

Mit dieser Methode wird eine veränderte Verbrauchs- oder Wetterprognose eines Agenten gespeichert.

Methode producedPowerOfDay

Methode consumedPowerOfDay

Mit diesen Methoden wird die produzierte bzw. konsumierte Leistung eines Agenten für einen Tag gespeichert.

Methode buyPower

Methode sellPower

Diese Methoden werden für jede gekaufte bzw. verkaufte Minute Leistung aufgerufen.

10.8.3. Klasse BBEvent

Die Klasse BBEvent enthält Methoden, mit denen das Blackboard seine Ereignisse in der Datenbank speichert. Die Ereignisse stehen alle im Zusammenhang mit Angeboten.

Methode addOffer

Methode removeOffer

Methode expireOffer

Diese Methoden werden aufgerufen, wenn es ein neues Angebot gibt, bzw. wenn ein bestehendes Angebot gelöscht wird, oder wenn ein Angebot abgelaufen ist.

Methode notifyNew

Methode notifyDel

Diese Methoden werden aufgerufen, wenn das Blackboard Agenten über neue bzw. gelöschte Angebote informiert.

10.8.4. Klasse `BREEvent`

Die Klasse `BREEvent` enthält eine Methode, mit der der BKV seine Bilanz für einen Tag in der Datenbank speichert.

Methode `accounting`

Diese Methode speichert den Energieüberschuß bzw. Energiemangel sowie die Einnahmen bzw. Ausgaben für einen Tag.

10.9. Paket `DezentenLib.Net`

Das Paket `DezentenLib.Net` beinhaltet Klassen, die die Kommunikation der Agenten über ein IP-Netzwerk ermöglichen und deren Details kapseln. Zur Serialisierung der Daten wird dabei auf die Java-Serialisierung mittels des Interfaces `Externalizable` zurückgegriffen. Alle Nachrichten werden in Instanzen der Klasse `Message` gekapselt. Die Kodierung und Dekodierung der Message-Objekte erfolgt durch die Klassen `Network` und `Receiver`.

Nachrichten werden durch Methodenaufrufe in der Klasse `Network` verschickt und dabei mit ihren Parametern in ein Message-Objekt gewandelt. Nach der Übertragung werden sie von einem Receiver empfangen. Dieser sorgt dann dafür, dass die entsprechende Methode im Empfängeragenten mit den übermittelten Parametern aufgerufen wird. Der Rückgabewert dieser Methode wird wieder in ein Message-Objekt verpackt und zurück an die Klasse `Network` übertragen, um senderseitig als Rückgabewert der aufgerufenen Methode an den Aufrufer zurückgegeben zu werden.

Zusätzlich ist in der Klasse `Message` hartcodiert, welche Nachrichtentypen synchron und welche asynchron behandelt werden. Im Fall von asynchronen Nachrichten sendet ein Receiver unmittelbar nach Empfang eine kurze Empfangsbestätigung zurück. Im anderen Fall wird, wie oben beschrieben, zuerst die entsprechende Methode aufgerufen und deren Rückgabewert zurückgeschickt.

10.9.1. Klasse `Network`

Die Klasse `Network` ist eine Hilfsklasse zur Netzwerkkommunikation und enthält nur statische Methoden. Die Methoden realisieren Kommunikati-

onsereignisse, d. h. für jeden Sendevorgang, z. B. wenn ein Verbraucher für ein Angebot eines Erzeugers bieten möchte, existiert eine spezielle Methode. Grundsätzlich blockieren die Methoden, bis eine Antwort vom Empfänger der Nachricht zurückgekommen ist (siehe auch `sendMsg`). Allerdings ist in einigen Fällen die Kommunikation der Agenten synchron in dem Sinne, dass mit Erhalt der Antwort das Kommunikationsereignis abgeschlossen ist, wie z. B. im Falle der Methode `recvOfferList`, die unmittelbar ihr Ergebnis, nämlich eine Angebotsliste des Blackboards, zurückliefert.

In anderen Fällen ist die Kommunikation asynchron in dem Sinne, dass ein Agent einen anderen über ein Ereignis informiert und dieser dann zu einem beliebigen späteren Zeitpunkt auf diese Information antwortet. Ein Beispiel hierfür wäre das bereits erwähnte Angebot eines Verbrauchers an einen Erzeuger (mittels der Methode `sendOfferToProducer`). Mit Erhalt des Angebots antwortet der Erzeuger nicht unmittelbar, sondern bestätigt zunächst nur den Erhalt der Nachricht und sendet erst zu einem späteren Zeitpunkt eine Zu- oder Absage auf das Angebot (mittels der Methode `sendOfferReply`).

Methode `sendMsg`

In dieser zentralen Methode findet der eigentliche Sendevorgang statt, allerdings wird diese Methode nur intern verwendet. Sie ist öffentlich deklariert, u. a. um Tests zu vereinfachen.

Die übergebene Nachricht wird versendet und anschließend wartet die Methode auf eine Antwort, die der entsprechende Receiver zu liefern hat. Die Antwortnachricht darf ein einzelnes Objekt enthalten, das dann an die aufrufende Methode zurückgeliefert wird.

10.9.2. Klasse `NetAddress`

Die Klasse `NetAddress` kapselt die Kombination von IP-Adresse und Port-Nummer. Sie wird unter anderem zur eindeutigen Identifizierung der Netzwerkschnittstelle eines laufenden Agenten verwendet. Da mehrere Agenten auf einem Rechner laufen können, reicht zur eindeutigen Identifizierung eines Agenten die IP-Adresse nicht aus.

10.9.3. Klasse `Message`

Um einzelne Nachrichten und speziell die Serialisierung und Deserialisierung von Objekten zu kapseln, wird die Klasse `Message` verwendet. Über die Konstruktoren können neue Nachrichten erstellt werden.

Methode `add`

Mit der Methode `add` können einer Nachricht beliebig viele Objekte hinzugefügt werden.

Methode `write`

Die Serialisierung eines Nachrichten-Objekts wird durch diese Methode realisiert. Alle zur Nachricht gehörenden (und mittels `add` hinzugefügten) Objekte müssen serialisierbar sein, d. h. hier, dass die entsprechenden Klassen bevorzugt das Interface `Externalizable` implementieren müssen. Auf diese Methode wird beim Senden von Nachrichten zurückgegriffen.

Methode `readMsg`

Für die Deserialisierung von Nachrichten, d. h. das Parsen des Datenstroms und der Erzeugung der so kodierten Objekte, ist diese Methode zuständig. Sie ist statisch und liefert ein Message-Objekt, das die deserialisierten Objekte enthält.

10.9.4. Klasse `Receiver`

Die abstrakte Klasse `Receiver` dient dazu, Nachrichten, die von der Klasse `Network` abgeschickt wurden, entgegenzunehmen und die entsprechende Methode in dem jeweiligen Agenten aufzurufen, dem der Receiver zugeordnet ist.

Dazu existiert innerhalb dieser Klasse eine innere Klasse `ReceiverListener`. Diese ist als Thread implementiert und wird bei der Erzeugung des Receiver-Objektes erzeugt und gestartet. Hier wird auf eingehende Verbindungen gewartet. Trifft eine Verbindung ein, wird ein weiterer Thread der inneren Klasse `ReceiverConnectionHandler` gestartet. Dieser ruft die dem empfangenen Message-Objekt entsprechende Methode auf, übergibt

die Parameter und sendet den Rückgabewert in einem neuen `Message`-Objekt an den Aufrufer zurück.

Für jeden Agententyp exisitiert eine eigene, von `Receiver` abgeleitete Klasse (`BalanceDistrictReceiver`, `BlackboardReceiver`, `Consumer-Receiver` und `ProducerReceiver`), die die von den jeweiligen Agenten implementierten und über das Netzwerk aufrufbaren Methoden kennt.

Konstruktor `Receiver`

Im Konstruktor der Klasse `Receiver` übergibt der Agent, in dem der Receiver erzeugt wurde, ein von `ReceiverDispatcher` abgeleitetes Interface – diese sind im einzelnen:

- `BalanceDistrictDispatcherTarget`
- `BlackboardDispatcherTarget`
- `ProducerDispatcherTarget`
- `ConsumerDispatcherTarget`
- `ControllerDispatcherTarget`

In diesem Interface sind die dem jeweiligen Receiver bekannten und über das Netz aufrufbaren Methoden definiert. Es wird ein TCP-Port geöffnet, auf dem der Receiver auf eingehende Nachrichten reagiert.

Methode `dispatchMessage`

Die Methode `dispatchMessage` wird von der Klasse `Receiver` aus aufgerufen, wenn eine Nachricht eintrifft. Sie besteht im Wesentlichen aus einer `switch`-Anweisung, in der je nach Typ der Nachricht die entprechende Methode im Interface `DispatcherTarget` aufgerufen wird. Diese Methode wird als einzige Methode von den spezialisierten Receivern überschrieben. Falls eine Nachricht eintrifft, deren Typ den spezialisierten Receivern nicht bekannt ist, wird die Methode der Superklasse aufgerufen. Falls der Typ auch hier nicht bekannt ist, wird eine Fehlernachricht zurückgesendet.

10.9.5. Interface `DispatcherTarget`

Dieses Interface wird als Basisinterface für die spezialisierten `DispatcherTarget`s verwendet. Es definiert die Methoden, die jeder Agent, der an der Kommunikation teilnimmt, implementieren muss.

Methode `startSimulation`

Diese Methode muss von allen Agenten implementiert werden und führt zum Start der Simulation, d. h. einem laufenden Agenten-Prozess wird auf diese Weise mitgeteilt, dass nun die eigentliche Simulation beginnt.

Methode `terminate`

Diese Methode muss von allen Agenten implementiert werden und führt zum Terminieren der Simulation und zum Beenden des jeweiligen Agenten. Damit kann eine auf mehreren Rechnern verteilt ablaufende Simulation gestoppt werden (siehe auch Abschnitt 10.10).

10.10. Paket `Simulation`

Alle für die Steuerung der Simulation und für die Verwaltung der Daten in der Datenbank notwendigen Klassen sind in diesem Paket angesiedelt. Das Frontend für die Bedienung ist von diesen Klassen getrennt und befindet sich im Paket `Simulation.GUI` (siehe auch Abschnitt 10.13 auf Seite 108).

Im folgenden sind die wichtigsten Klassen des Pakets `Simulation` aufgelistet:

`ConfigDB`

`ControllerDispatcherTarget`

`Controller`

`DezentenUI`

`RegisteredAgentManager`

`SimulationStartupHandler`

Die Ausführung des Programms zur Simulationskontrolle beginnt in der Klasse `DezentenUI`. Dort werden Instanzen der Klasse `Controller` und `Simulation.GUI.MainFrame` erzeugt und miteinander verknüpft. Die Klasse `Controller` koordiniert die Datenflüsse und Ereignisse, die bei der Steuerung der Simulation auftreten.

Eine weitere Aufgabe ist, die Agenten der Simulation (letztlich mittels einer *Secure Shell*) verteilt zu starten. Dazu wird die Klasse `Simulation-StartupHandler` verwendet. Im Gegenzug kümmert sich die Klasse `RegisteredAgentManager` um die Beendigung der gestarteten Prozesse.

Wie für die Agenten existiert auch für den Controller eine Spezialisierung der Klasse `DezentenLib.Net.Receiver`, nämlich die Klasse `Controller-Receiver`. Außerdem wird durch den Controller das `ControllerDispatcherTarget` implementiert, so dass er wie die Agenten auch an der Netzwerkkommunikation der Simulation teilnehmen kann.

Über die oben beschriebene Schnittstelle rufen die Simulationsteilnehmer ihre jeweils benötigten Konfigurationsdaten (z. B. Wetterdaten, Kennlinien oder Netzwerkadressen) ab. Diese wiederum erhält der Controller aus einer Datenbank, auf die er über statische Methoden der Klasse `ConfigDB` Zugriff erhält.

10.11. Das Datenbank-Modell

Die Tabellen des Modells (siehe C auf Seite 191) lassen sich grob in zwei Gruppen unterteilen, nämlich den Tabellen für die Konfiguration einer Simulation sowie den Tabellen zur Aufnahme der Ergebnisse eines Simulationslaufes.

Die zentrale Tabelle für die Konfiguration ist `config` und dient zur eindeutigen Identifizierung eines Konfigurationsdatensatzes. Zu einer Konfiguration gehören im Wesentlichen eine Auswahl von Agenten (`agentset`) sowie die den Agenten zugeordneten Kennlinien (`ap_rel`). Zusätzlich wird die Tabelle `hg_rel` benötigt, um mehrere Rechner in einer Gruppe zusammenzufassen, so dass Simulationsteilnehmer automatisch verteilt auf diesen Rechnern gestartet werden können.

Die während eines Simulationslaufes anfallenden Daten werden in folgenden Tabellen gespeichert:

`prognosis:` korrigierte Wetterprognosen

power: produzierte und konsumierte Energie

accounting: Ergebnis der Abrechnung der verhandelten Energie

offer: alle jemals aufgetretenen Angebote

offerevent: Ereignisse, die in Zusammenhang mit Angeboten stehen

bre_acc: Energiebilanz des Bilanzkreises

Mit Hilfe der Tabelle `simulation` werden all diese Daten eindeutig einem Simulationslauf zugeordnet.

10.12. Paket `Simulation.DbWrapper`

Die folgenden in diesem Paket enthaltenen Klassen dienen als einfache Wrapper für Datenbank-Objekte:

Curve

Hostgroup

Host

SimulationConfig

Es soll eine Beziehung zwischen einer Datenbank-ID und dem jeweiligen in dem Wrapper gekapselten Objekt hergestellt werden.

10.13. Paket `Simulation.GUI`

Die Kompontenten der graphischen Oberfläche sind in diesem Paket zusammengefasst. Das Hauptfenster der Anwendung ist in der Klasse `MainFrame` realisiert. Es besteht aus mehreren Panels, die in einer `JTabbedPane` angeordnet sind. Die einzelnen Panels sind voneinander unabhängig und greifen nur auf Dienste des Controllers zurück. Von signifikanten Ereignissen in jeweils anderen Panels erfahren sie über die Ereignisbehandlung des Controllers.

Jedes Panel ist in einer eigenen Klasse implementiert:

StartupControlPanel: Einstellungen zum Start einer Simulation

ConfigurationPanel: Erstellung von Konfigurationen und Agenten

HostgroupPanel: Erzeugung von Rechnernamen und -gruppen

AgentAssignPanel: Zuordnung von Agenten zu Konfigurationen

AgentCurveAssignPanel: Zuordnung von Kurven zu Agenten

CurvePanel: Design von Kurven

EvaluationPanel: Auswertungen

OnlineEval: Online-Auswertung des Blackboards

10.14. Paket Technics

Das Paket Technics enthält die folgenden Klassen:

Technics Abstrakte Basisklasse, die nur gemeinsame Datenstrukturen sowie ein einfaches Interface zur Verfügung stellt.

ConsumerTechnics Verwaltet die Technik des Verbrauchers, bietet eine Schnittstelle zur Prognosekorrektur und erledigt die Abrechnung.

ProducerTechnics Verwaltet die Technik des Erzeuger-Agenten, bietet eine Schnittstelle zur Prognosekorrektur und erledigt die Abrechnung.

AbstractPrognosis Basisklasse zur Prognosebehandlung und -korrektur.

LinearPrognosis Implementation der linear approximierenden Prognosekorrektur.

MovAvgPrognosis Implementation der Prognosekorrektur mittels gleitendem Durchschnitt.

SimplePrognosis Klasse, die keine Änderungen an der Prognose vornimmt.

10.14.1. Klasse `Technics`

Neben einigen gemeinsamen Datenstrukturen enthält die Klasse `Technics` die abstrakte Methode `generatePrognosis`, die als einheitliche Schnittstelle für die Agenten dient. Vor jedem Abruf einer Prognose durch einen Agenten ist vorher diese Methode aufzurufen.

10.14.2. Klasse `ConsumerTechnics`

Es werden sowohl reale als auch prognostizierte Verbrauchskurven verwaltet, bei Bedarf Prognosekorrekturen erstellt und auch wesentliche Teile der Abrechnung durchgeführt.

Die Verbraucher-Technik teilt sich die zentrale Datenstruktur mit dem Verbraucher-Agent.

Methode `addInterval`

Mit Hilfe dieser Funktion werden die Datenstrukturen, die den realen Bedarf des Verbrauchers speichern, aktualisiert.

10.14.3. Klasse `ProducerTechnics`

Die Klasse `ProducerTechnics` wird benötigt, um dem Producer-Teil des Agenten die Leistung, die gerade produziert wird bzw. die prognostiziert wurde, mitzuteilen.

Methode `setParamsFromStrategy`

Die Methode `setParamsFromStrategy` setzt aus der Strategie des Agenten die Parameter für die Berechnung der neuen Prognose. Die Parameter sind:

- die Abweichung, welche festlegt, wie stark eine Abweichung der Prognose vom realen Verlauf höchstens sein darf
- die Zeitdauer, für die in die Zukunft hinein die Prognose korrigiert werden soll
- der Glättungsparameter, der angibt, wie stark die Glättung ausfallen soll

Methode getPrognosis

Diese Methode dient ausschließlich dazu, eine Kopie der Datenstruktur zu liefern, die die prognostizierte Produktion für die nächsten 24 Stunden enthält.

Methode addContract

Die Methode addContract meldet der Technikklasse einen abgeschlossenen Vertrag. Die Methode wird vom Agenten aufgerufen und dient als Hilfsmethode zur Abrechnung.

Methode doAccounting

Die Methode doAccounting rechnet ab. Die Abrechnung benutzt die Information über den realen Verlauf des Wetters und über die abgeschlossenen Verträge. Dazu werden die Methoden setProduction, getDesiredProduction und getProduction benutzt.

Methode setProduction

Die Methode setProduction setzt die Produktion für die Abrechnungperiode fest. Das ist ein einfacheres und übersichtlicheres Verfahren, als nach jedem Vertragsabschluß die Produktion zu setzen.

Methode getDesiredProduction

Die Methode getDesiredProduction gibt die Soll-Produktion, die durch die Methode addContract festgelegt wurde, für einen Tag zurück.

Methode getProduction

Die Methode getProduction gibt die produzierte Leistung, die durch die Methode setProduction erstellt wurde, für einen Tag zurück.

10.14.4. Klasse AbstractPrognosis

Die in der Klasse AbstractPrognosis definierte abstrakte Methode doCorrection ist der Einstiegspunkt für die Prognosekorrektur. Abgeleitete

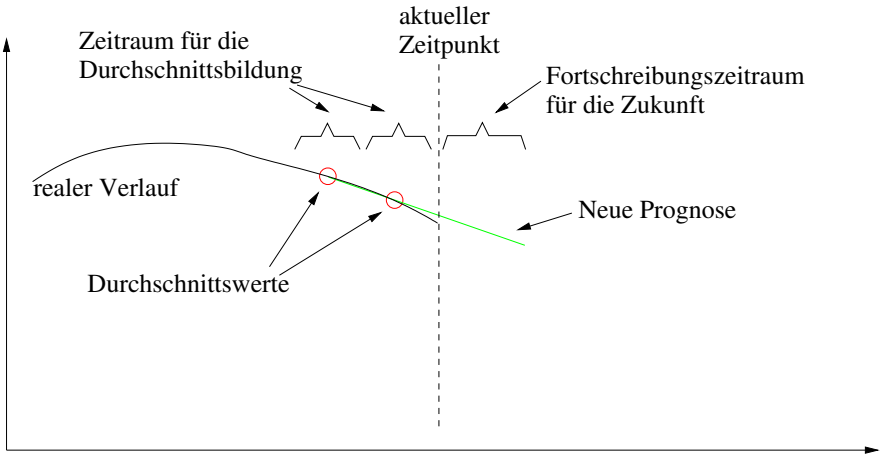

Abbildung 10.1.: Funktionsweise der linearen Prognosekorrektur.

Klassen, die jeweils einen konkreten Korrekturalgorithmus implementieren, stellen unter anderem die folgenden Methoden bereit:

isUpdateNeeded um festzustellen, ob die Abweichung der bisherigen Prognose so groß ist, daß eine Korrektur nötig wird

doCorrection um die eigentliche Korrektur durchzuführen

Das Ergebnis dieser Korrektur verwaltet die Klasse `AbstractPrognosis`. Mit der Methode `getValue` können die ursprünglichen bzw. korrigierten Prognosewerte abgerufen werden.

10.14.5. Klasse `LinearPrognosis`

Diese Klasse implementiert einen linear approximierenden Algorithmus zur Korrektur von Prognosen. Dabei wird der reale Kurvenverlauf der unmittelbaren Vergangenheit betrachtet – dieser Zeitraum wird halbiert, für jedes Intervall der Durchschnitt gebildet und durch diese beiden Werte eine Gerade gelegt, die die Prognose für die unmittelbare Zukunft beschreibt, siehe Abb. 10.1. Die konkreten Intervalllängen sind parametrisiert und können zur Laufzeit variiert werden.

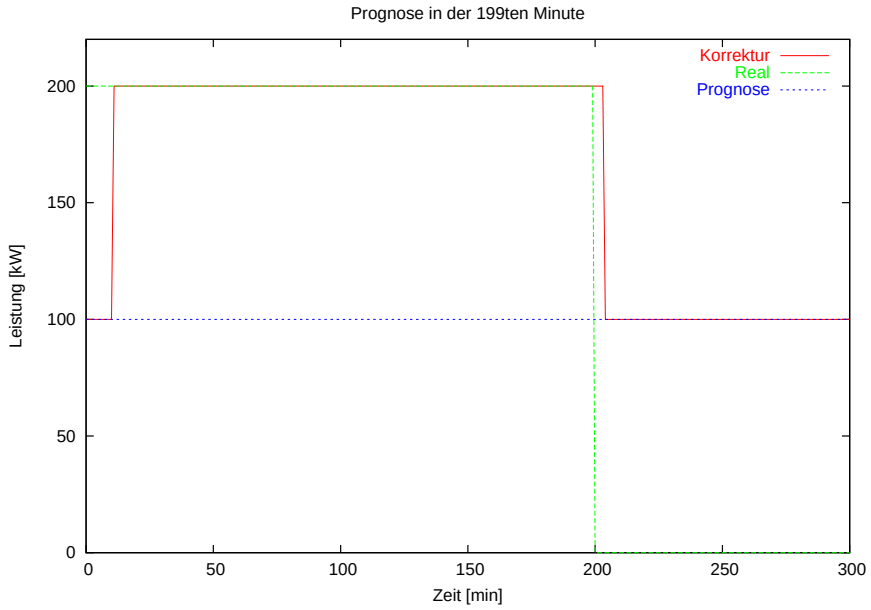

Abbildung 10.2.: Prognose in der 199. Minute.

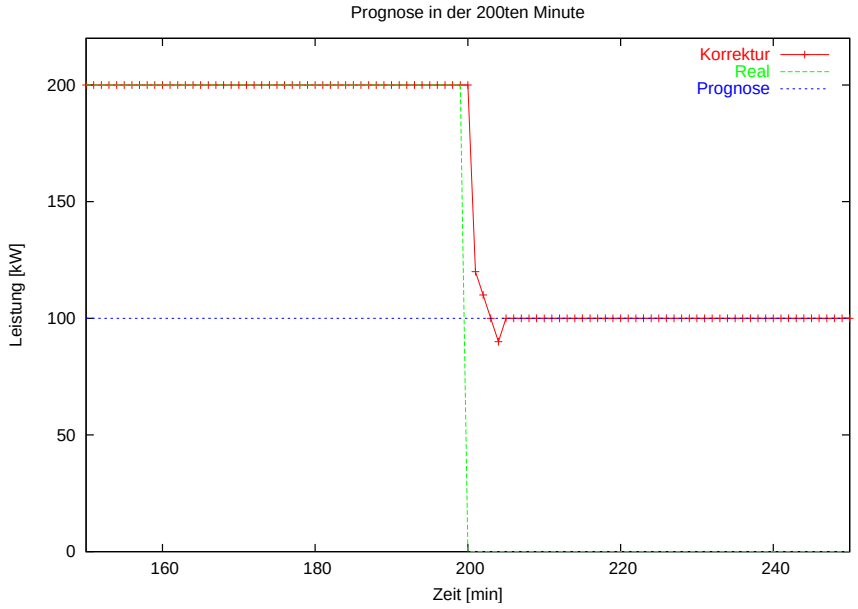

Abbildung 10.3.: Prognose in der 200. Minute.

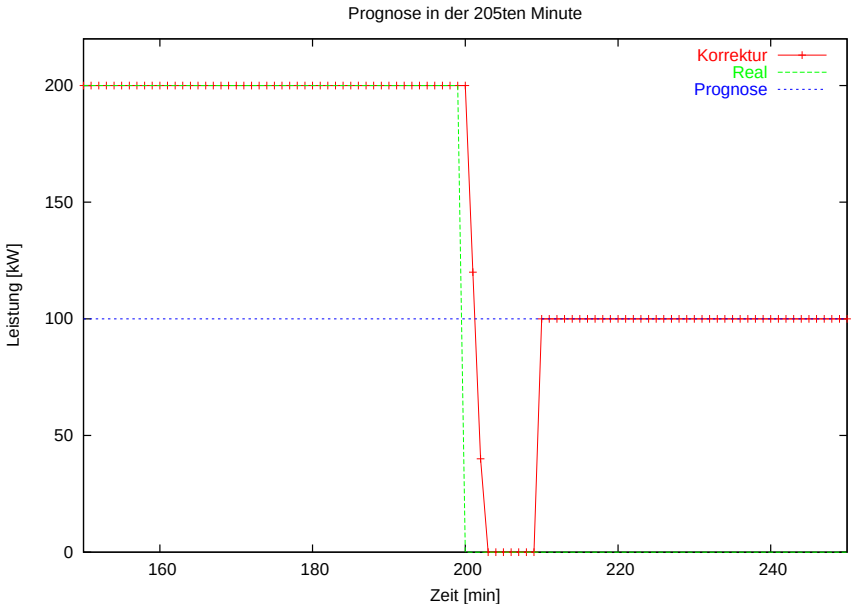

Abbildung 10.4.: Prognose in der 205. Minute.

Im folgenden betrachten wir ein Beispiel für diesen Algorithmus. In der 200. Minute gibt es einen Abfall des Energieverbrauches auf null. Die Prognose geht von einem gleichmäßigen Verbrauch von 100 für den ganzen Tag aus, siehe Abb. 10.2. Nach den ersten 10 Minuten (nach Simulationsstart) hat die Korrekturfunktion genügend Werte für eine Anpassung und stellt sich auf einen Verbrauch von 200 ein. Auch in der 199. Minute (direkt vor dem Ausfall) wird die Prognose fortgeschrieben. Hieran zeigt sich, warum der Zeitraum, für den korrigiert wird, nicht zu groß gewählt werden sollte. Denn für den Bereich zwischen der roten und grünen Kurve wird eingekauft, eine Minuten später wird ein Teil dieser Energie jedoch wieder zum Verkauf freigegeben, da sich die Korrekturkurve an den realen Verlauf angepaßt hat, Abb. 10.3.

Ab der Minute 205 ist die Korrekturfunktion aus dem Knick heraus und korrigiert die Prognose für die nächsten fünf Minuten völlig korrekt auf 0 herunter. Danach gilt wieder der alte Verlauf, Abb. 10.4.

Der Algorithmus läßt sich durch drei Parameter beeinflussen. Die fol-

gende Tabelle zeigt die Zuordnung zu Strategienamen (wie sie auch in der GUI auftauchen) und den Einzelparametern.

Strategie	deviation	deviationrange	correctionrange
linear short	10	5	10
linear long	20	10	10

deviation die Abweichung (in %) der vorgegebenen Prognose vom realen Verlauf. Überschreitet die Abweichung diesen Wert nicht, so findet keine Korrektur statt. Dieser Wert liegt für gewöhnlich bei 10 bis 20 %.

deviationrange der Zeitraum der Vergangenheit, der betrachtet werden soll. Dieser sollte mindestens 10 und maximal 30 Minuten betragen. Je größer der Wert, desto langsamer die Reaktion/Beeinflussung durch kleine Sprünge.

correctionrange der Korrekturzeitraum für die Zukunft. Er sollte zwischen fünf und zehn Minuten betragen. Größere Werte sind nicht sinnvoll, da insbesondere bei Anstiegen (bei Abfällen kann die Prognose nur bis auf 0 herabsinken) sonst zu große Werte erreicht werden.

10.14.6. Klasse `MovAvgPrognosis`

Diese Klasse realisiert den Algorithmus des gleitenden Durchschnitts[1]. Die Methode `doCorrection` ist der Einstiegspunkt und kümmert sich um die Verwaltung einiger Hilfsdatenstrukturen. Die eigentliche Implementation findet sich in der Methode `correctPrognosis`, die sich einiger weiterer Hilfsmethoden bedient.

Die eigentliche Korrekturmethode ist `invert`. Bei ihr geht man davon aus, dass die Werte in der Zukunft tendentiell von Werten aus der Vergangenheit abhängen (Trend des Wetters). Die Werte der Vergangenheit

[1] Der gleitende Durchschnitt kann als gewichtete Summe über eine bestimmte Anzahl von n Werten aufgefasst werden:

$$s_t = \sum_{j=-\tau}^{\tau} w_j y_{t+j} \quad \text{für } t = \tau+1,\cdots,n-\tau$$

s_t ist dann der gleitende Durchschnitt $2p+1$-er Ordnung, wobei y_t die Realisierung der Zufallsvariable Y zum Zeitpunkt t ist und $w_j = \frac{1}{2\tau+1}$.

werden also am Leistungswert bei t_0 invertiert, d.h. alle Werte von t_0 bis t_{-range} werden bei t_0 einmal vertikal und einmal horizontal gespiegelt.

Wie gut ist die Prognosekorrektur mit diesem Algorithmus? Die folgende Tabelle zeigt, wie sich die Unterschiede der Parameter in den willkürlich benannten Strategien auswirkt:

Strategie	τ (tau)	deviation	range
medium	100	100	120
no risk	50	50	30
dangerous	100	200	300
mixture	75	130	180

τ (tau) der Glättungsfaktor. Welche Auswirkung er auf die Korrektur hat, kann man am besten sehen, wenn man sich den Unterschied zwischen dem realen und dem prognostizierten Verlauf der Leistung in Abb. 10.6 ansieht. Die Verlaufskurve der Leistungsprognose wurde, bis auf die Abweichung, aus der Verlaufskurve der tatsächlichen Leistungserzeugung – mit der Methode des gleitenden Durchschnitts – erstellt.

deviation die minimale Abweichung, ab der die Korrektur aktiviert wird. Bei kleineren Abweichungen gibt es keine Korrektur. Zu beachten ist, dass in Abb. 10.6 die Deviation aller Strategien auf 100 gesetzt wurde. In den Experimenten entsprechen die Werte der obigen Tabelle. Hier liessen sich die Stratiegien, wegen des daraus resultierenden unterschiedlichen Einsetzens nur schwerer vergleichen. Trotzdem erkennt man das die unterschiedlichen Parameter einen Einfluss auf die Korrektur haben.

range die Weite der Korrektur in die Zukunft. Allerdings wird auch noch eine Angleichung an die ursprüngliche Prognose, die für 24 Std. im vorraus galt, durchgeführt. Das heißt, die tatsächliche Weite ist:
`range + weite_bis_ursprüngliche_prognose =`
`range + (x_weite(Abweichung - Schnittpunkt) * k)`
wobei $k =]0, 2]$, je nach dem ob der Schnittpunkt[2] in der Zukunft mit der ursprünglichen Prognose gefunden wird.

[2] In dem Korrekturmodell geht man davon aus, das bei der Abweichung in t_0, also beim aktuellen Zeitpunkt, die Korrektur beginnt. Es wird angenommen, dass unmittelbar in der Vergangenheit es einen Schnittpunkt der Prognosekurve mit der Wetterkurve

Bei der Angleichung der korrigierten Prognose an die ursprüngliche geht man – wie bei `invert` – davon aus, dass sich die Änderung der Prognose genau so schnell angleicht wie sie auch entstanden ist. Somit wird der tendentielle Verlauf der Abweichung zwischen dem Schnittpunkt in der Vergangenheit und t_0 als Verlauf für die Angleichung benutzt. Er wird also vertikal an t_0 gespiegelt und bei t_{range} angehängt.

Damit die Prognose nicht geringfügige Abweichungen (Zacken der Abb. 10.5) anzeigt, wird die Methode der gleitenden Durchschnitte darauf angewendet um die Zacken zu glätten.

Die Abb. 10.5 zeigt den ursprünglichen Zustand, d.h. die Prognose und den realen Verlauf der Leistung und Abb. 10.6 den Zustand nach der Korrektur:

gegeben hat und dass nach der Korrektur, die beiden Kurven sich in einem zukünftigen Schnittpunkt wiederfinden.
Wird der Schnittpunkt in der Zukunft nicht gefunden, dann ist das nicht weiter schlimm. Es gibt dann lediglich einen Sprung der korrigierten Prognose zur ursprünglichen. Wird aber der Schnittpunkt in der Vergangenheit der Prognosekurve mit der Wetterkurve nicht bis $t_{Simulationsbegin}$ gefunden, bricht der Algorithmus ab.

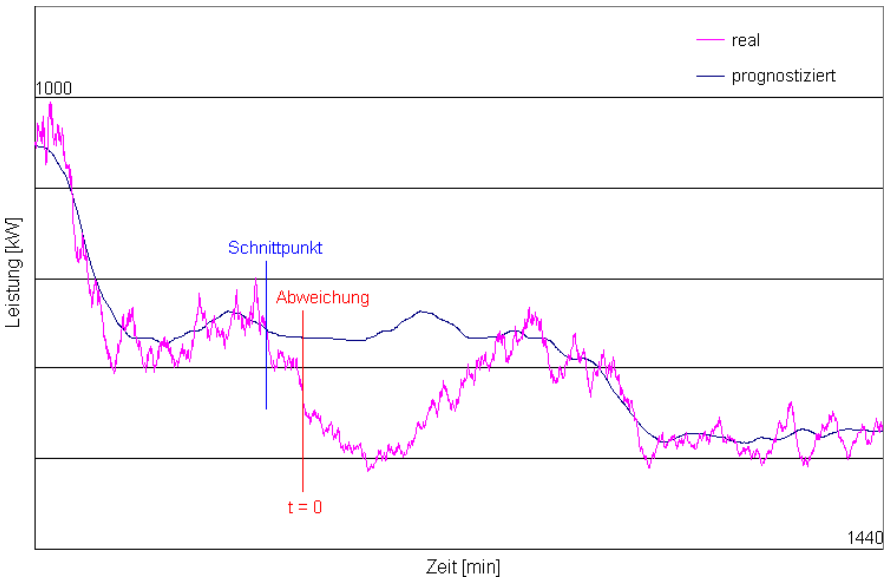

Abbildung 10.5.: Prognose und realer Verlauf der Leistung.

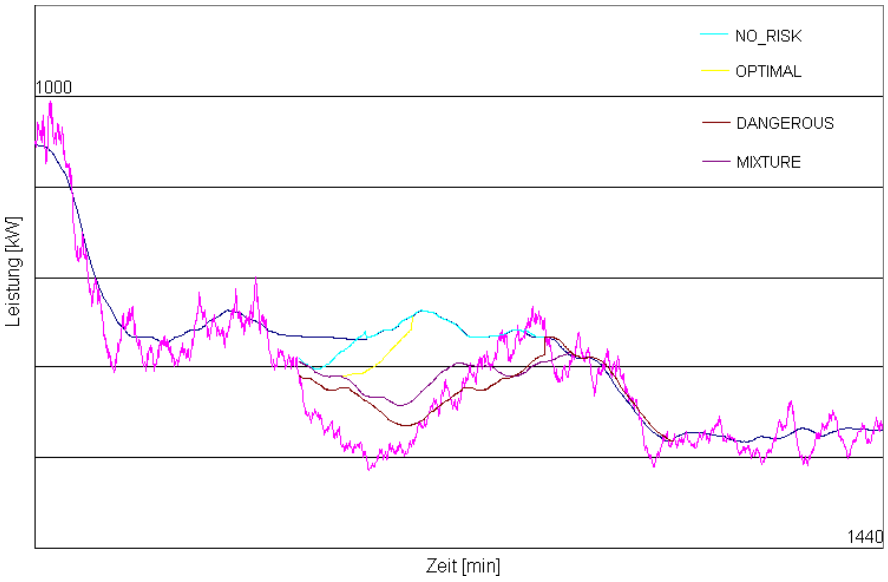

Abbildung 10.6.: Prognose nach der Korrektur

11. Bedienungsanleitung

Das „Dezenten User Interface“ (DUI) ist die grafische Steuerzentrale der Simulationsumgebung DEZENTEN. Von hier aus lassen sich alle wesentlichen Dinge der Simulation konfigurieren.

Nach dem Start präsentiert sich dem Anwender ein Hauptfenster das eine Menüleiste und mehrere Karteikartenreiter enthält.

Das Hauptmenü stellt unter dem Menüpunkt *File* eine Funktion zum Exportieren der Konfiguration (siehe 11.10) und zum Beenden des Programms zur Verfügung. Der Menüpunkt *Settings* erlaubt das Verändern der Arbeitsumgebung (siehe 11.1.1) und der Datenbankeinstellungen (siehe 11.1.2). Unter *Help* befindet sich der Menüpunkt *About*, der ein Informationsfenster mit dem Namen des Programms und der Versionsnummer anzeigt.

11.1. Allgemeine Einstellungen

Bei der erstmaligen Benutzung des DUI sind die Daten der Arbeitsumgebung und der Datenbank einzustellen. Die Abb. 11.1 zeigt Grundeinstellungen im Computerpool des Fachbereiches Informatik der Universität Dortmund.

11.1.1. Arbeitsumgebung

Allgemein gilt, daß alle Pfade, die hier eingestellt werden, auf allen an der Simulation teilnehmenden Rechnern gleich sein müssen.

Path to classes Der Pfad zu den Klassen der Simulationsumgebung.

Path to start script Der Pfad zum Startscript, welches die Agenten startet.

Path to log files In diesem Verzeichnis werden Logdateien angelegt, welche eventuelle Fehlermeldungen einzelner Teilsysteme aufnehmen können.

Debug level Hier kann eingestellt werden, welche Arten von Meldungen in den Logdateien gespeichert werden.

- *fatal* Es werden nur Fehler protokolliert, die eine weitere Ausführung verhindern.
- *error* Es werden nur schwerwiegende Fehler protokolliert.
- *warn* Es werden schwerwiegende Fehler und Warnungen protokolliert.
- *info* Es werden zusätzlich zu [warn] noch allgemeine Informationen ausgegeben.
- *debug* Es werden zusätzlich zu [info] eine Menge interner Informationen ausgegeben, z.B. Datenstrukturen.

Trace Output Wenn diese Option eingeschaltet ist, werden verschiedene Einträge, die sonst nur in die Datenbank geschrieben werden, auch auf der Konsole ausgegeben.

11.1.2. Datenbankeinstellungen

JDBC-Driver Der zur Verbindung mit der Datenbank eingesetzte Treiber. Er sollte bei der Nutzung der Datenbank mySql auf `com.mysql.jdbc.Driver` eingestellt sein.

Host Der Name des Rechners, der die Datenbank bereitstellt.

Port Der TCP/IP Port der Datenbank auf obigem Rechner.

Database Der Name der Datenbank.

Username Der Name eines Benutzers, der in der Datenbank eingetragen ist.

Password Das zum Datenbanknutzer dazugehörige Passwort.

Es sei darauf hingewiesen, daß die einzelnen Teilprozesse auf den entfernten Rechnern per `ssh` gestartet werden. Daher müssen für alle teilnehmenden Computer gültige rsa-Schlüssel vorliegen, sowie die automatische Passwortauthentifizierung aktiviert sein.

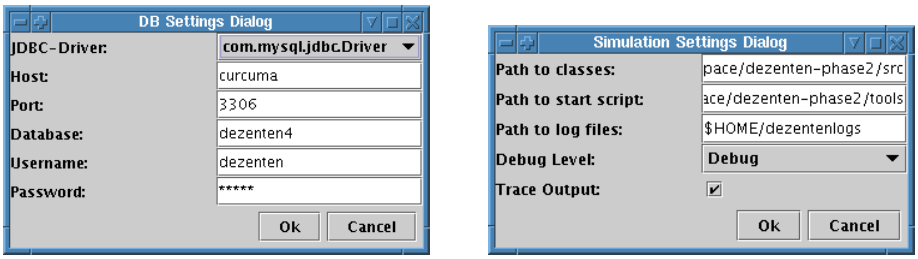

Abbildung 11.1.: Dialoge zur Grundeinstellung der Datenbank und der Arbeitsumgebung.

11.2. Startup Control

Das Fenster *Startup Control* erlaubt es, festzulegen, welche Agentengruppe (Configuration) auf welchen Computern (Hostgroup, siehe 11.4) an einer Simulation teilnehmen. Desweiteren setzt man hier die Startzeit der Simulation fest. Der Beschleunigungsfaktor gibt an, um welchen Faktor die Zeit in der Simulation schneller läuft als die wirkliche Zeit, ein Wert von „1“ bedeutet, daß keine Beschleunigung stattfindet. Ein Wert von „20“ bewirkt, dass eine Minute in 3 Sekunden echter Zeit simuliert wird.

Die Simulation wird folgendermaßen gestartet:

Start agents Dieser Knopf erzeugt einen neuen Eintrag in der Datenbank für die ausgewählte Kombination an Agenten. Der aktuelle Simulationslauf erhält dabei eine eindeutige Nummer unter der er immer wieder abgefragt werden kann. Anschließend werden die Agenten auf den beteiligten Computern gestartet, die danach im unteren Feld aufgelistet werden.

Start simulation Dieser Knopf gibt den Agenten das Signal, daß sie mit den Verhandlungen beginnen können.

Terminate simulation Dieser Knopf beendet alle Agenten und damit auch die Simulation.

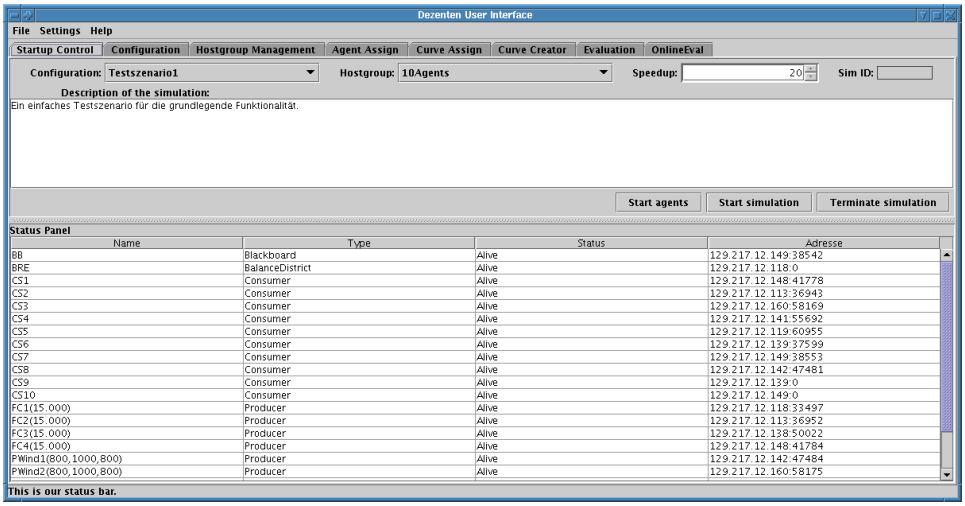

Abbildung 11.2.: Das Fenster *Startup Control* ermöglicht die Auswahl der Agentengruppe und der Rechnergruppe sowie Start und Ende der Simulation.

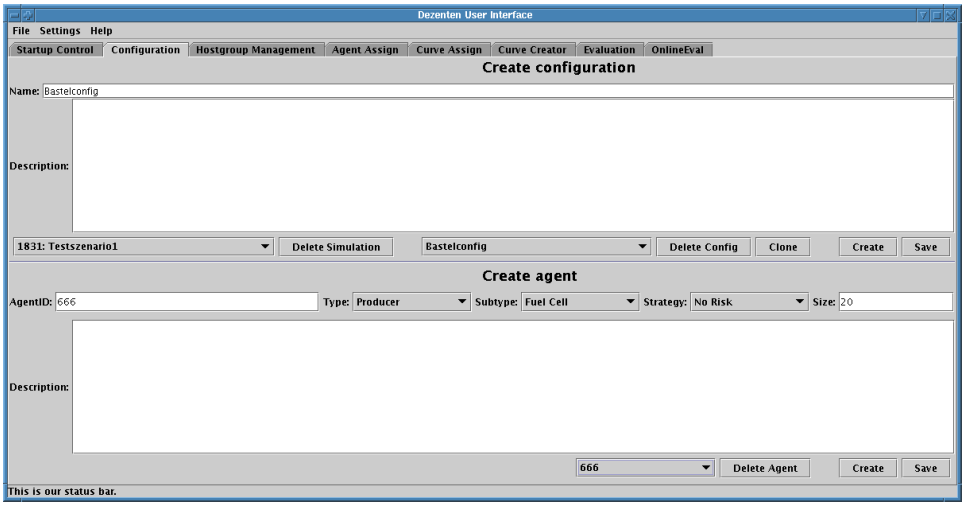

Abbildung 11.3.: Die *Configuration* erlaubt das Anlegen neuer Szenarien und Agenten.

11.3. Configuration

Das Fenster *Configuration*, siehe Abb. 11.3, erlaubt es, neue Simulationsszenarien, Konfigurationen genannt, und neue Agenten zu erzeugen.

Im oberen Bereich können neue Konfigurationen angelegt werden. Dazu ist lediglich ein Name und eine textuelle Beschreibung einzugeben. Durch den Button create wird das neue Szenario in die Datenbank eingefügt. Danach können dieser Konfiguration Agenten zugewiesen werden, siehe Abschnitt 11.1.2. Darüber hinaus können Konfigurationen gelöscht, geklont und gespeichert werden.

Im unteren Bereich des Fensters können neue Agenten angelegt werden. Dazu ist ein neuer Name und der Typ des Agenten (Black Board, Balance Responsible Entity, Consumer, Producer) anzugeben. Bei den Erzeugern (Producer) ist zudem der Erzeugertyp (Solarzelle, Windrad, Brennstoffzelle) anzugeben. Bei einer Brennstoffzelle muß zusätzlich die Kapazität angegeben werden.

Wird als Typ „Producer“ ausgewählt, muß zusätzlich die Strategie festgelegt werden. Diese kann aus der Dropdown-Liste ausgewählt werden. Eine Übersicht der Strategien und ihrer Korrekturverfahren zeigt Tabelle 11.1. Es stehen folgende Strategien zur Verfügung:

none Falls keine Strategie gewählt wird, werden die Parameter des Producers auf Standardwerte gesetzt.

dangerous Die Strategie *dangerous* ist eine riskante Angebotserstellungsstrategie. Hier wird nur eine minimale Abweichung nach unten von der prognostizierten Menge und dem kalkulierten Preis gestattet. Der Preis wird in allen Strategien mittels eines Algorithmus aus den Ankaufs- und Verkaufspreisen des Bilanzkreises sowie den Parametern der bereits am Blackboard angebotene Angebote berechnet. Die Preisanpassung erfolgt hier so, daß der Preis nach der Angebotserstellung zunächst nur geringfügig sinkt und erst kurz vor dem Lieferzeitpunkt des Angebots stark reduziert wird.

no risk Die Strategie *no risk* ist eine Strategie, bei der die Wahrscheinlichkeit, daß der Producer Verlust erleidet, minimiert wird. Hervorzuheben ist, daß die Mengen und Preise eines Angebotes mit einem

Algorithmus in der Form berechnet werden, daß sie in vielen Fällen deutlich unter den prognostizierten Werten liegen, um somit die Wahrscheinlichkeit eines nötigen Nachkaufes zu minimieren. Außerdem verläuft die zeitliche Preisanpassung der Angebote derart, daß der Preis schon früh relativ stark gesenkt wird, um die Angebote wenigstens mit etwas Gewinn verkaufen zu können.

medium Bei der Strategie *medium* wird die Kombination der Parameter so gesetzt, daß der Erwartungswert des Gewinns, den der Producer durch sein Handeln erreicht, maximiert wird. Das wird dadurch erreicht, daß die zeitliche Preisanpassung linear erfolgt und die Mengen und Preise eines Angebotes mit einem Algorithmus in der Form berechnet werden, daß sie immer knapp (maximal 10%) unter den prognostizierten Werten liegen.

linear short Eine Kombination der Strategie *medium* mit der linearen Korrekturprognose.

linear long Wie auch bei *linear short* wird die lineare Korrekturprognose verwendet. Allerdings reagiert *linear long* etwas langsamer, da die Abweichung über einen doppelt so langen Zeitraum existieren muß.

Strategie	Prognosekorrektur	Schwellenwert[a]	Korrektur-zeitraum[b] [min]
none	CORRECT_NONE	–	–
medium	CORRECT_MOVE_AVG	60 absolut	60
no risk	CORRECT_MOVE_AVG	30 absolut	20
dangerous	CORRECT_MOVE_AVG	100 absolut	120
linear short	CORRECT_LINEAR	10 %	10
linear long	CORRECT_LINEAR	20 %	10

[a] Dieser Werte bezeichnet die nötige Abweichung der bisher gültigen Prognose vom realen Verlauf bevor korrigiert wird.
[b] Der Zeitraum den die Prognose in die Zukunft reicht.

Tabelle 11.1.: Strategien und ihre Prognosekorrekturen.

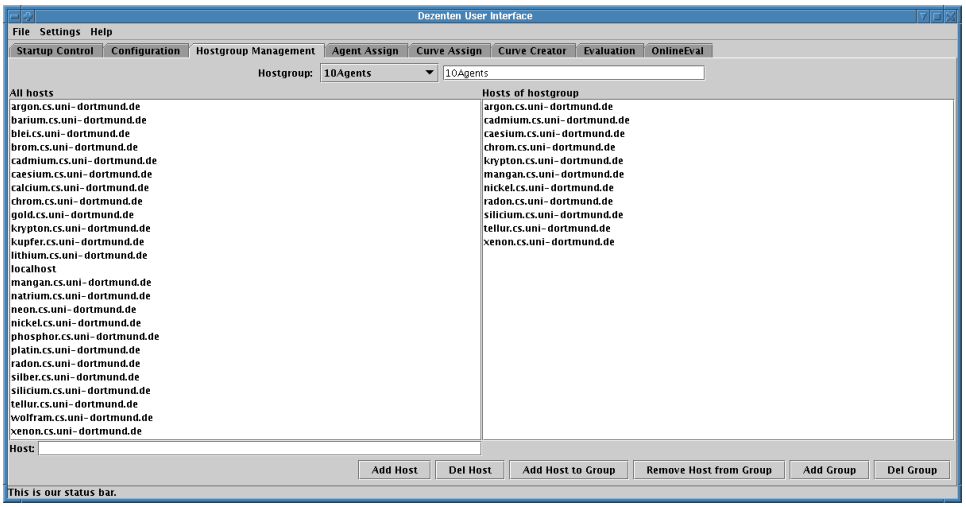

Abbildung 11.4.: Das *Hostgroup Management* erlaubt das Zusammenfassen von Computern zu Gruppen.

11.4. Hostgroup Management

Da es sich bei DEZENTEN um ein verteiltes System handelt, sind in der Regel mehrere Computer an einem Simulationslauf beteiligt. Daher können in diesem Fenster verschiedene Computer, die an einem Experiment teilnehmen (hosts) zu Gruppen zusammengefaßt und in eine Konfiguration eingefügt werden, siehe Abb. 11.4.

Die Dropdown-Liste oben in der Mitte ermöglicht die Auswahl einer Gruppe. Das linke Fenster zeigt alle verfügbaren Computer, das rechte Fenster zeigt alle Computer, die in der aktuell ausgewählten Gruppe enthalten sind.

Die Schaltflächen ermöglichen das Hinzufügen und Löschen von Computern, von Gruppen und von Computern zu bzw. aus Gruppen.

11.5. Agent Assign

Hier können zu einer bestehenden Konfiguration, die aus der Dropdown-Liste oben in der Mitte des Fensters *Agent Assign* ausgewählt werden kann, Agenten zugeordnet werden.

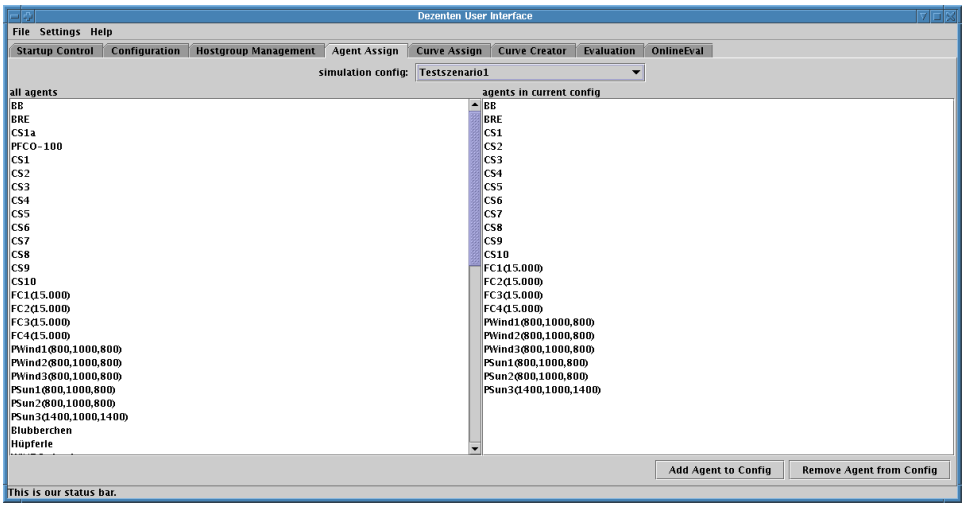

Abbildung 11.5.: Das Fenster *Agent Assign* erlaubt das Entfernen oder Hinzufügen von Agenten aus bzw. zu einer bestehenden Konfiguration.

Auf der linken Seite befindet sich eine Liste der Agenten, rechts werden die Agenten angezeigt, die bereits in die aktuell bearbeitete Konfiguration eingefügt wurden. Wird ein Agent in der linken Liste angeklickt, kann er mit Hilfe des Buttons *Add Agent to Config* zur Konfiguration hinzugefügt werden. Analog lassen sich Agenten aus der Konfiguration mit dem Button *Remove Agent from Config* entfernen.

11.6. Curve Assign

Das Fenster *Curve Assign* bietet die Möglichkeit, einem Agenten eine oder mehrere Kennlinien zuzuordnen. Diese kann z.B. eine Prognosekurve sein. Im linken Teil des Fensters werden alle vorhandenen Kurven aufgelistet. Weitere Kurven können mit Hilfe des Fensters *Curve Creator* (siehe Abschnitt 11.7) erstellt werden. Diese Kurven bzw. Kennlinien können im rechten Teil des Fensters einem ausgewählten Agenten zugeordnet werden. Auswählen kann man den Agenten in der rechten Dropdown-Liste, die sich im oberen Teil befindet. Links daneben kann in einer weiteren Dropdown-Liste die Konfiguration selektiert werden.

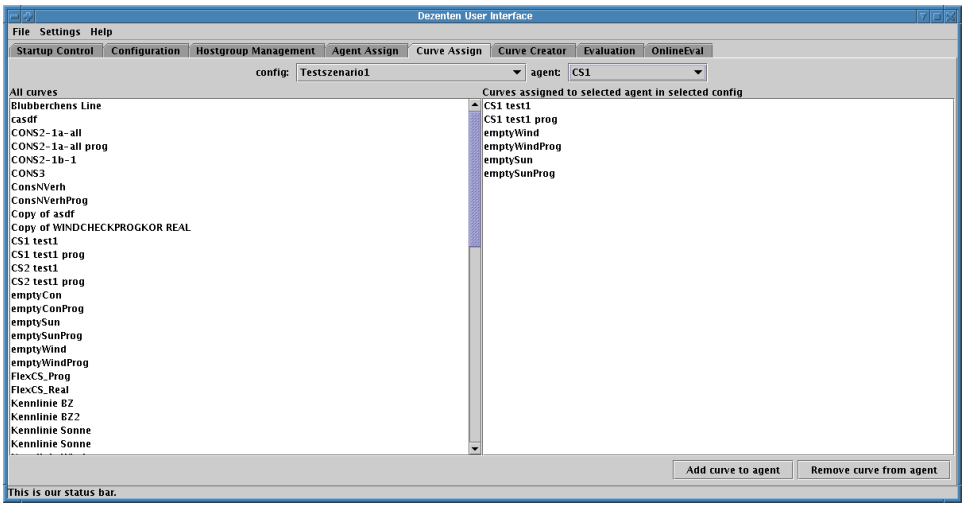

Abbildung 11.6.: Im Fenster *Curve Assign* werden Agenten Kennlinien zugeordnet.

11.7. Curve Creator

Mit Hilfe des Fensters *Curve Creator*, siehe Abb. 11.7, kann man vorhandene Kennlinien verändern, klonen, entfernen oder völlig neue Kennlinien erstellen. Im linken Teil des Fensters befindet sich eine Liste aller verfügbaren Kurven. Sobald eine Kurve angeklickt wird, erscheint ihr Verlauf rechts daneben. Weiter unten gibt es die Möglichkeit, die ausgewählte Kurve zu bearbeiten. Der Name, der minimale und maximale X-Wert (die Zeit)[1], sowie Grad des Polynoms. Bei Verbrauchern sind zusätzlich Intervallstart und -stop zu setzen. Diese sind die Grenzen, in denen Verbrauch und Realer Verlauf vom Consumeragenten verschoben werden dürfen (flexibler Verbraucher).

Beschreiben den Verbrauchers können verändert werden. Dabei geben Intervallstart und -stop die Anfangs- und Endminute des jeweiligen Intervalls an.

Außerdem kann aus einer Dropdown-Liste der Typ ausgewählt werden. Im rechten Teil des Fensters befinden sich Buttons zum Erstellen, spei-

[1]Im Fall von Verbraucher-Kurven muss der minimale X-Wert der Kurven 0 betragen, damit er sich in der Minute 0 initialisiert.

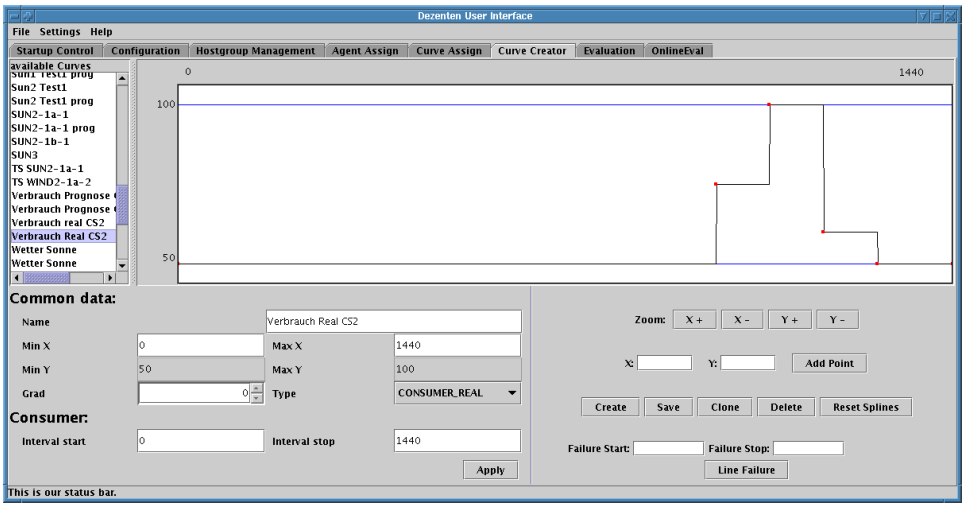

Abbildung 11.7.: Der *Curve Creator* erlaubt das Erstellen und Verändern von Kennlinien.

chern, klonen, entfernen und zurücksetzen einer Kennlinie, basierend auf den Daten, die links eingegeben wurden.

Darüber hinaus kann man den Verlauf einer Kurve auch direkt verändern. Durch Klicken im rechten Teil des Fensters können weitere Verlaufspunkte hinzugefügt werden. Alternativ dazu können einzelne Punkte auch gesetzt werden, indem ihre X- und Y-Koordinate in die im rechten Teil des Fensters dafür vorgesehenen Felder eingetragen werden und dann der Button *Add Point* gedrückt wird. Vier Buttons erlauben das Zoomen in X- und Y-Richtung.

Rechts unten können Leitungsausfälle simuliert werden. Dazu gibt man ein Intervall an, innerhalb dessen die Leistungskurve auf 0 gesetzt wird. Dies entspricht z. B. einem Erzeuger, der keinen Strom liefert.

11.8. Evaluation

Das Fenster *Evaluation*, siehe Abb. 11.8, bietet die Möglichkeit, sich nach Abschluss der Simulation offline statistische Daten anzeigen zu lassen. Diese Daten werden aus der Datenbank ausgewertet.

Im Bereich *Simulation* kann man hierfür die gewünschte Simulations-

nummer eingeben und den Knopf *Update Agents* drücken. Dadurch wird die Dropdown-Liste *AgentID*, in der sich die Agenten-IDs und der Eintrag *all* befindet, der stellvertretend für alle Agenten-IDs steht, aktualisiert.

Im Bereich *Time Ranges* kann mit *from* und *until* der Zeitraum festgelegt werden, für den die Auswertungen durchgeführt und angezeigt werden sollen. Soll eine Prognose angezeigt werden, so muß ein genauer Zeitpunkt angegeben werden, da sich die Prognosen in die Zukunft erstrecken und es daher (durch Nachkorrektur) für einen einzelnen Zeitpunkt mehrere Prognosewerte gibt. Dies ermöglicht das Feld *Prognosis Snapshot*.

Der Bereich *Gnuplot Options* erlaubt das Einstellen des Linientypen in den angezeigten Diagrammen. Zur Verfügung stehen: einfache Linien, Linien bei denen die einzelnen Punkte markiert sind und Balkendiagramme.

Die Anzeige der Daten erfolgt durch folgende Knöpfe:

Bought Power Hier kann man sich die gekaufte Leistung in Form einer Kurve anzeigen lassen.

Sold Power Die verkaufte Leistung wird in Form einer Kurve ausgegeben.

Consumed/Bought Power dient zur Erstellung des Diagramms für die verbrauchte und gekaufte Leistung. Beide Kurven werden in einem Diagramm dargestellt.

Produced/Sold Power Die produzierte und verkaufte Leistung des ausgewählten Agenten wird ausgegeben. Auch hier sind beide Kurven in einem Diagramm zu sehen.

Agent Info zeigt sowohl die produzierte, verkaufte, verbrauchte und eingekaufte Leistung eines ausgewählten Agenten an

Billing zeigt die Rechnungssumme des ausgewählten Agenten an, dazu gehört die Rechnungssumme des Agenten mit dem Bilanzkreis sowie die Summe der Rechnungen mit den anderen Agenten.

Price Development stellt die Preisentwicklung in Cent für den angegebenen Zeitraum dar.

Prognosis Curves zeigt die Prognose, die realen Daten und die Korrektur der Prognose für einen Agenten an.

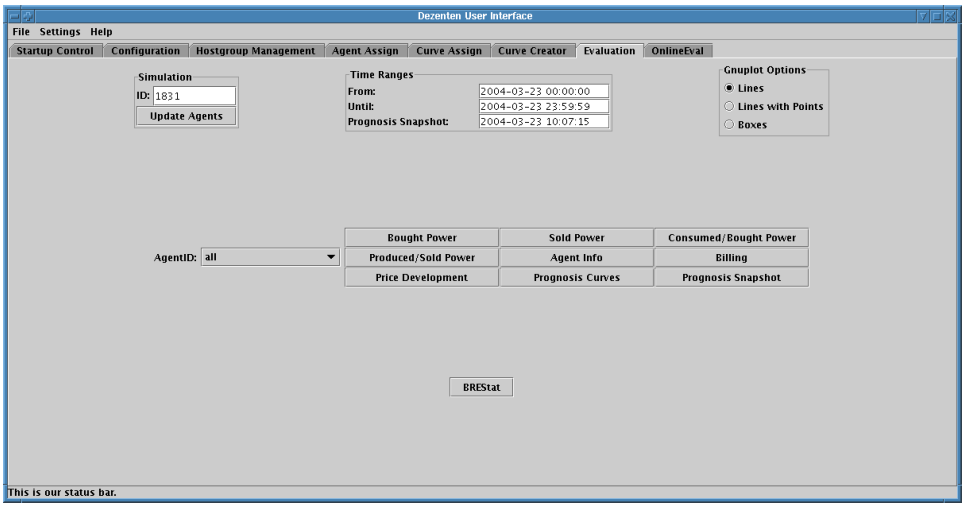

Abbildung 11.8.: Das Fenster *Evaluation*, welches zur Statistikausgabe dient

Prognosis Snapshot zeigt die gekaufte, die verkaufte, die reale, die prognostizierte Leistung und die Prognosekorrektur eines ausgewählten Agenten an.

Unten in der Mitte des Fensters befindet sich ein weiterer Button, der zum Ausgeben der Statistik des Bilanzkreises dient. Diese Statistik enthält die vom Bilanzkreis aufgekaufte Leistung, die vom Bilanzkreis verkaufte Leistung sowie deren Differenz.

11.9. OnlineEval

Im Fenster *OnlineEval* des DUI ist es während der Simulation möglich, zu verfolgen, welche Aktionen am Blackboard passieren. Es bietet eine Online-Auswertung der Angebote, die am Blackboard stehen, neu eingestellt und entfernt werden. Neben den Angeboten sind auch Minimal-, Maximal-, Durchschnittspreis, sowie die Gesamtleistung und die Simulationszeit aufgeführt. Rechts unten kann eingestellt werden, wie oft das Fenster aktualisiert werden soll.

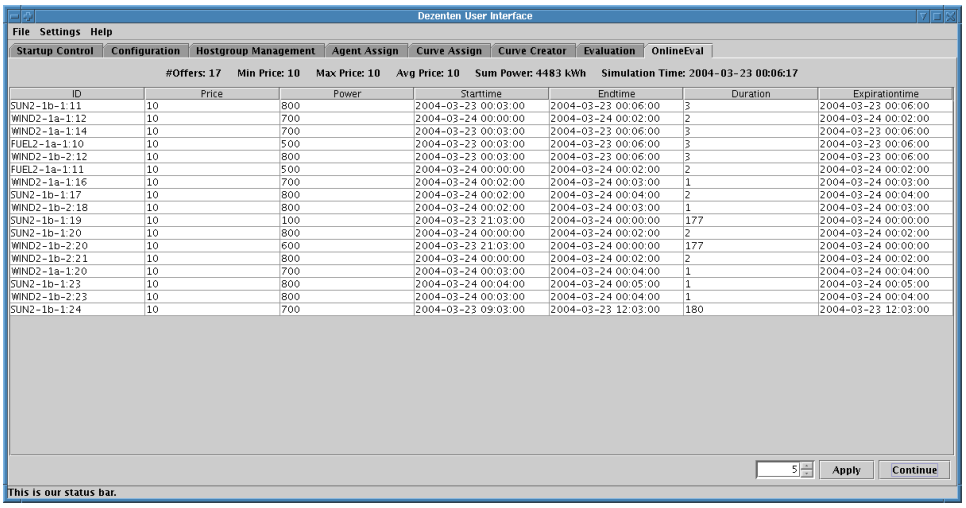

Abbildung 11.9.: Das Fenster *OnlineEval* dient zur Online-Auswertung. Im Bild sind einige eingestellte Angebote zu sehen.

11.10. Export der Konfiguration

Die in der Datenbank gespeicherte Konfiguration kann durch Auswählen des Menüpunktes Export im Menü File in eine Datei gespeichert werden. Es werden nur die für die Konfiguration wichtigen Daten exportiert, die bei den Testläufen angefallenen Daten werden nicht mit exportiert. Die Konfigurationsdaten werden in Form von SQL-Statements gespeichert, mit denen die Konfiguration in eine neue Datenbank importiert werden kann. Dazu müssen zuvor in der neuen Datenbank die Tabellenstruktur angelegt werden und dann die SQL-Statements in der durch die Export-Funktion erstellten Datei ausgeführt werden. Je nach verwendeter Zieldatenbank müssen noch die automatisch generierten Schlüssel auf den richtigen Startwert initialisiert werden.

12. Integration, Simulationsläufe und Auswertung

Die Tests wurden in drei größere Szenarien unterteilt, die die wesentlichen Funktionalitäten des Programms testen sollten. Innerhalb dieser Szenarien gibt es bis zu drei Unterfälle, die speziellere Funktionen prüfen. Die drei Testszenarien sind im einzelnen:

- Testszenario Eins (Abschnitt 12.1) testet die grundsätzlichen Funktionalitäten der Agenten. Dabei soll im ersten Unterfall die angebotene Energie von den Agenten komplett aufgekauft werden, da ihr jeweiliger Bedarf mit ihrer Prognose übereinstimmt. Im zweiten Unterfall werden die flexiblen Verbraucher getestet.
- Testszenario Zwei (Abschnitt 12.2) prüft die Nachverhandlung und die Prognosekorrektur. Wesentlich ist die Auswirkung der Prognosekorrektur auf das Ergebnis. Deshalb werden zwei Unterfälle mit der gleichen Konfiguration gestestet, einer mit Prognosekorrektur und einer ohne. Im letzten Unterfall wird der Leistungsausfall bei einzelnen Agenten simuliert.
- Testszenario Drei (Abschnitt 12.3) testet die verschiedenen Strategien der Erzeguer. Dabei soll heraus gefunden werden, unter welchen Bedingungen welche Strategie das optimale Ergebnis liefert.

12.1. Testszenario Eins

In diesem Testszenario werden die grundsätzlichen Funktionalitäten der Agenten getestet. Dazu wurden zwei Testfälle konstruiert. Im ersten Testfall wird das BlackBoard, der BRE, die verschiedenen Produzenten und der unflexiblen Verbraucher getestet. Der flexible Verbraucher kann in diesem Testfall nicht getestet werden, da der Verbrauch dieses Typs nicht

genau vorhersagbar ist. Die Höhe des Verbrauchs ist zwar bekannt, jedoch ist der Zeitpunkt des Verbrauchs verschiebbar. Deshalb wird der flexible Verbraucher in dem zweiten Testfall gesondert getestet.

12.1.1. Testfall 1

Erwartungen

Es nehmen alle Sorten von Agenten an diesem Test teil, d. h. Blackboard, BRE, Agent als Verbraucher und Agent als Produzent (Solarzelle, Brennstoffzelle und Windkraftrad). Die Verbraucher nehmen an diesem Test nur als unflexible Verbraucher teil, da bei einem flexiblen Verbraucher der Zeitpunkt des Verbrauchs nicht genau vorhersagbar ist; für diese Art von Test ist dies aber unbedingt notwendig, da sich die Summe der benötigten und erzeugbaren Energie genau ausgleicht. Die Agenten sollen miteinander sicher kommunizieren und verhandeln. Als Ergebnis dieses Tests wird erwartet, dass die Produzenten ihre produzierte Leistung komplett verkauft haben bzw. dass die Verbraucher diese produzierte Leistung von den Produzenten kaufen. Die Agenten haben in diesem Testszenario keinen Bedarf nachzuverhandeln, da die Prognosen und der tatsächliche Bedarf identisch sind. Dies hat natürlich auch zur Folge, dass keine Prognosekorrektur durchgeführt wird. Zudem darf auch keine Energie vom BRE gekauft bzw. an den BRE verkauft werden, da ausreichend Energie in unserer Simulation zur Verfügung steht. Die Strategien sollen keinen Einfluss auf den Test haben. Deshalb haben alle Agenten die Strategie *medium*, die die default-Strategie ist. Diese und alle weiteren Strategien sind im Kapitel 10.3 erklärt.

Konfiguration

Es nehmen 22 Agenten teil, davon sind zehn Agenten Verbraucher mit einem Bedarf von insgesamt 240.000 kW.[1] Ein Agent ist das Blackboard, ein weiterer der BRE. Die restlichen zehn Agenten sind Produzenten (vier

[1] Die Leistungsangaben wurden gewählt, um glatte Beträge zu erhalten. Dass einige gewählte Werte für bestimmte Akteure unrealistisch sind, ist nicht weiter problematisch; entscheidend ist ein sinnvolles Verhältnis der Leistungsangaben der Agenten untereinander

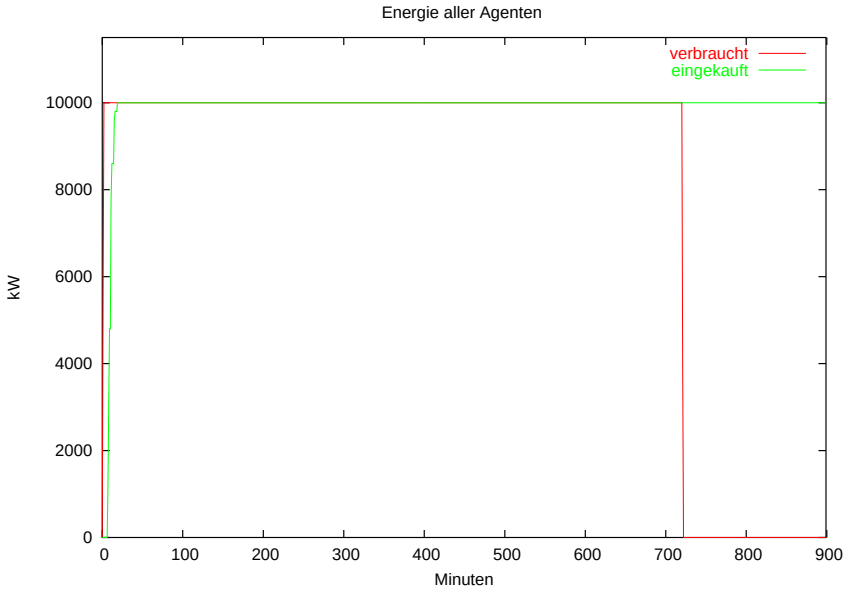

Abbildung 12.1.: Die gekaufte Energie aller Agenten (Sz. 1 Fall 1)

Brennstoffzellen mit insgesamt 96.000 kW, drei Solarzellen mit insgesamt 72.000 kW und drei Windkrafträder mit insgesamt 72.000 kW). Der Verbrauch ist der Produktion so angepasst, dass zu jedem Zeitpunkt der Bedarf von den Verbrauchern allein durch die Produzenten gedeckt werden kann. Die Produzenten haben als Strategie *medium*, siehe Kapitel 10.3, die Verbraucher haben eine entsprechende Strategie.

Ergebnisse

Die Auswertung des Tests ergab, dass die Verbraucher die erzeugte Energie der Produzenten aufgekauft haben, siehe Abbildungen 12.1 und 12.2.

Bei der großen Menge an Energie, die verhandelt wurde, stellte sich jedoch heraus, dass die Agenten einige Minuten Anlaufzeit brauchen, bis sie ihren Bedarf vollständig decken konnten. Allerdings ist er ab dann auch komplett abgedeckt.

Dieser kurze Einschwingvorgang ist für ein System mit (theoretisch) unbegrenzter Laufzeit nicht weiter problematisch. Er tritt zudem nur dann

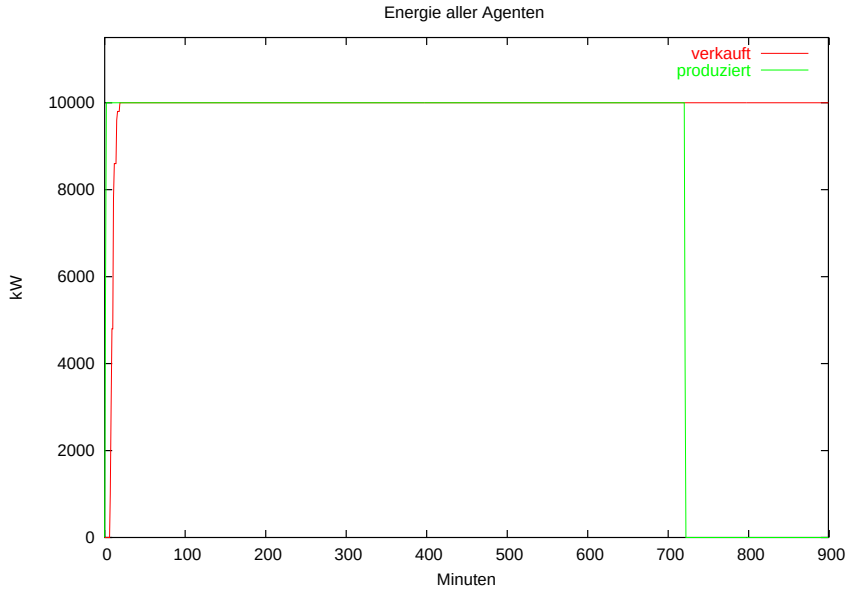

Abbildung 12.2.: Die verkaufte Energie aller Agenten (Sz. 1 Fall 1)

auf, wenn die Agenten in den ersten Minuten besonders viel Energie kaufen müssen. In Abbildung 12.3 ist dies gut zu erkennen.

12.1.2. Testfall 2

Erwartungen

In diesem Testfall wird getestet, ob die flexiblen Verbraucher funktionieren. Es nehmen zwei flexible Verbraucher teil, die ihre Verbrauchsintervalle so legen sollen, dass sie jeweils genug Energie von den Produzenten einkaufen können.

Konfiguration

Es nehmen zwei flexible Verbraucher und ein Produzent teil. Der Produzent ist eine Brennstoffzelle, die konstant 500 kW produziert. Die beiden flexiblen Verbraucher haben jeweils ein 20-minütiges Verbrauchsintervall, das sie innerhalb einer Stunde verbrauchen müssen. Die Leistung, die sie in diesem Interval brauchen, ist konstant 500 kW.

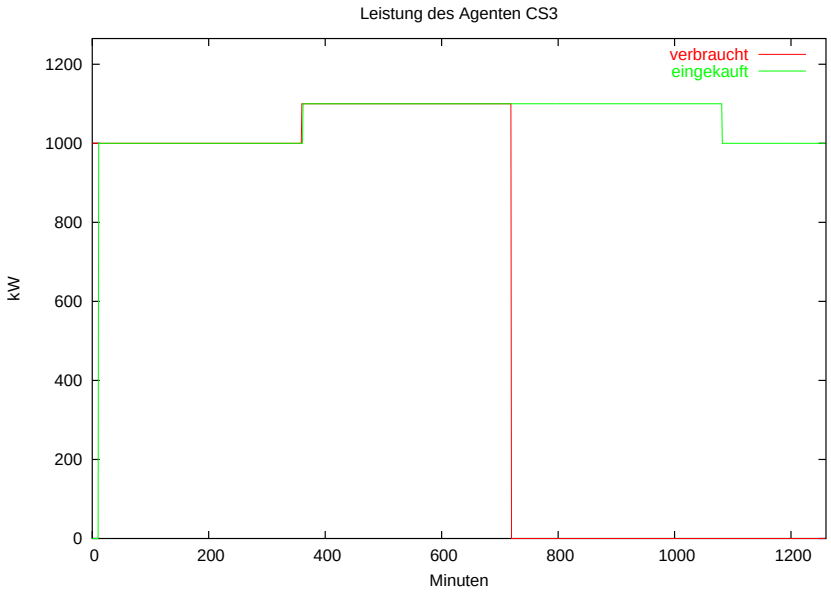

Abbildung 12.3.: Die gekaufte Energie des Verbrauchers CS3 (Sz. 1 Fall 1)

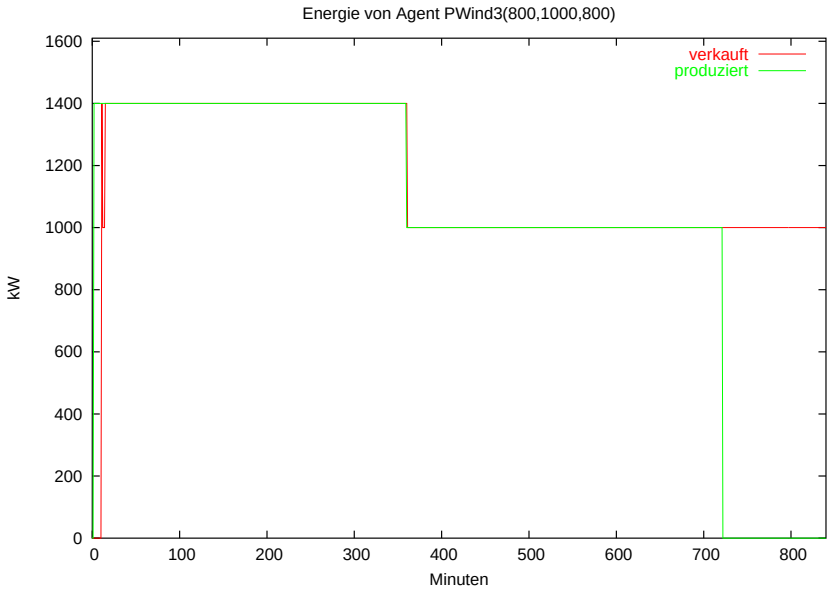

Abbildung 12.4.: Beispielverlauf für eine Abweichung bei PWind3 (Sz. 1 Fall 1)

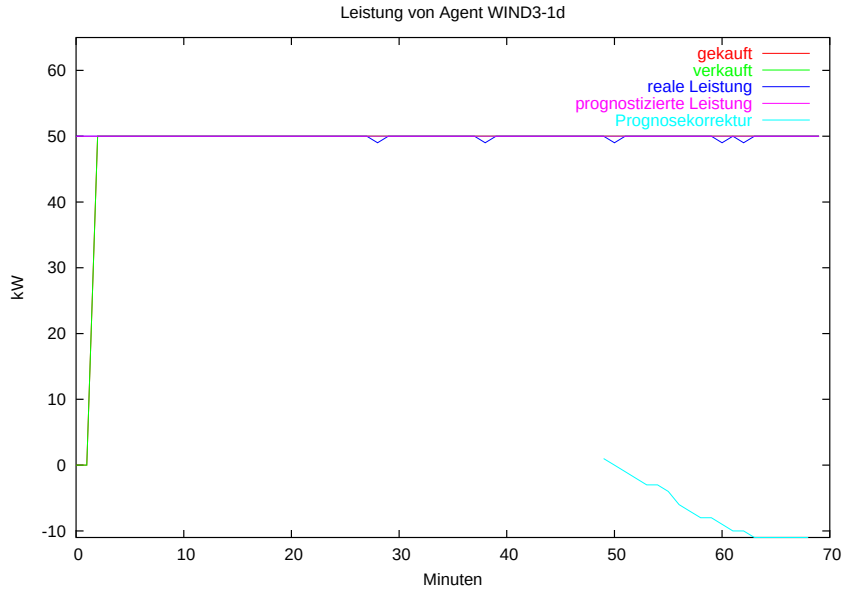

Abbildung 12.5.: Die gekaufte Energie des Verbrauchers flexConsumer1 (Sz. 1 Fall 2)

Ergebnisse

In den Diagrammen 12.5 und 12.6 ist klar zu sehen, dass die Verbraucher ihre Verbrauchsintervalle auf unterschiedliche Positionen geschoben haben. Somit kommen sie mit der produzierten Energie aus und brauchen keine Energie vom Bilanzkreis einzukaufen. In Abbildung 12.7 sieht man, dass der Produzent in diesem Test zweimal für 20 Minuten Leistung produziert. Die beiden Intervalle stimmen mit den Verbrauchsintervallen der flexiblen Consumer überein.

12.2. Testszenario Zwei

Nachdem im ersten Testszenario die grundlegenden Funktionalitäten getestet wurden, werden in diesem Testszenario die Nachverhandlungsmöglichkeit und die Prognosekorrektur der Simulation integriert.

Dazu sollen auch die Auswirkungen der Prognosekorrektur in dieser Konfiguration getestet werden. Es werden jeweils zwei Testdurchläufe mit glei-

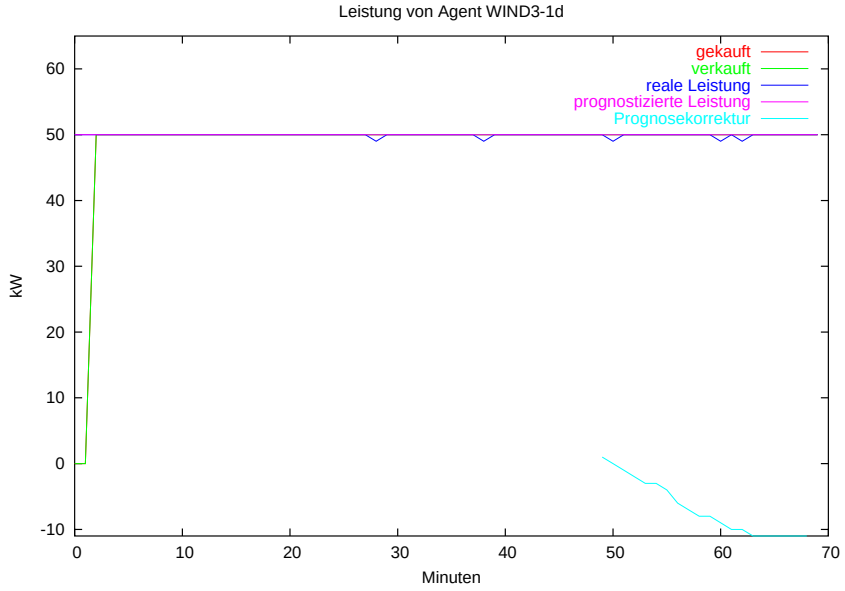

Abbildung 12.6.: Die gekaufte Energie des Verbrauchers flexConsumer2 (Sz. 1 Fall 2)

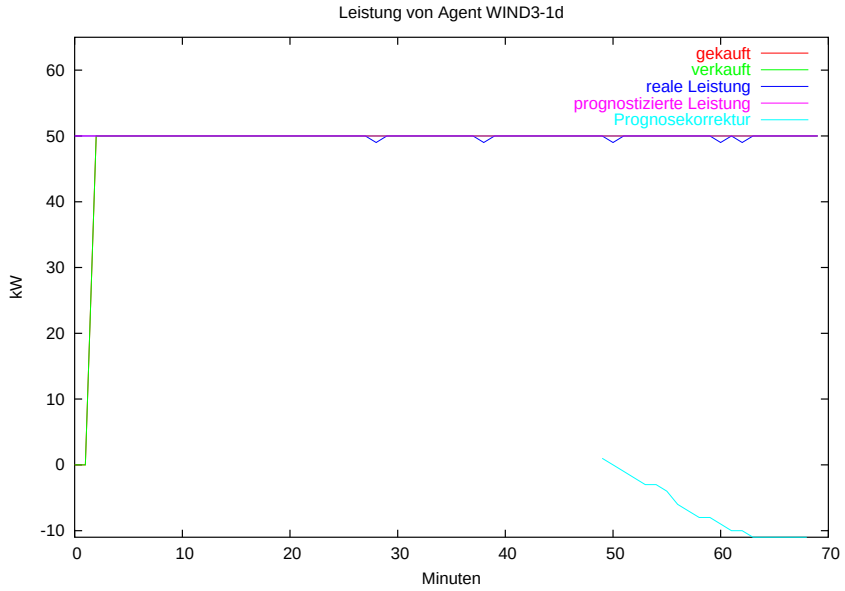

Abbildung 12.7.: Die gekaufte Energie aller Agenten (Sz. 1 Fall 2)

chen Konfigurationen durchgeführt, wobei der erste Lauf ohne, der zweite mit Prognosekorrektur und der damit verbundenen Nachverhandlung stattfindet. Durch den Vergleich der beiden Testläufe soll erkannt werden, inwieweit sich die mögliche Prognosekorrektur und die Nachverhandlung positiv auf das Ergebnis auswirken. Dazu ist anzumerken, dass die Agenten nur dann nachverhandeln können, wenn die Technik die Prognose korrigiert.

Die Agenten hätten nur die Möglichkeit, noch nicht verkaufte Angebote, die von der Korrektur betroffen sind, zu ändern. Da sich die Prognosekorrektur immer auf die nächsten Minuten, vielleicht auch Stunden bezieht, die Agenten aber Angebote je nach Strategie für mehrere Stunden an das Blackboard stellen, sind in der Regel betroffene Angebote bereits verkauft. Die Agenten könnten also nur über eine Nachverhandlung auf die Prognoseänderung reagieren.

Das Testszenario Zwei unterteilt sich in drei Testfälle:

- Im Testfall eins wird die Prognosekorrektur und Nachverhandlung nicht ermöglicht. Dabei gibt es im Unterfall a) zwei Produzenten, die Abweichungen von Prognose und realem Verlauf aufweisen. Analog weisen im Unterfall b) ein Produzent und ein Verbraucher Abweichungen auf.
- Der Testfall zwei hat eine identische Datenkonstallation wie Testfall eins, wobei die Agenten die Möglichkeiten zur Prognosekorrektur und zur Nachverhandlung haben.
- Im Testfall drei wird abschließend der Leitungsausfall bei einzelnen Agenten simuliert.

In diesem Szenario sollen die Agenten gezwungen werden nachzuverhandeln, indem die Prognosen von den realen Werten abweichen. Die Abweichungen sollen alle möglichen Fälle abdecken, d. h., dass

- die Leistungsprognosen leicht über bzw. unter der realen Leistung liegen.
- die Leistungsprognosen stark über bzw. unter der realen Leistung liegen.

- in der Simulation sowohl Energiemangel als auch Energieüberschuss lokal bei den Agenten auftreten, so dass sie gezwungen sind, beim BRE nachzukaufen bzw. an ihn zu verkaufen oder mit anderen Agenten zu verhandeln.

Die Grundkonfiguration der einzelnen Testfälle ist identisch. Da bereits im ersten Testszenario gezeigt wurde, dass die Simulation mit 22 Agenten einwandfrei funktioniert, beschränkt sich dieser Fall aus Gründen der einfacheren Konfiguration, der besseren Übersicht und der leichteren Auswertung nun auf zehn Agenten. Neben dem Blackboard und dem BRE nehmen vier Produzenten und vier Verbraucher teil. Von den Produzenten sind zwei vom Typ Wind, im folgenden WIND1 und WIND2 genannt, einer vom Typ Sun, im folgenden SUN genannt und einer vom Typ Fuel, im folgenden FUEL genannt. Die Verbraucher sind unflexibel, d. h. sie haben fest vorgegebene Intervalle, in denen sie Energie benötigen. Diese Unflexibilität ermöglicht es, die Produzenten und Verbraucher so aufeinander abzustimmen, dass die Energie, die angeboten wird, tatsächlich auch genau nachgefragt wird. Alle Verbraucher haben eine Verbrauchsprognose von durchgängig 700 kW. WIND1 liefert, wie es auch die Prognose vorhersagt, 700 kW. WIND2 weist im realen Verlauf Abweichungen von der Prognose auf, ist aber so konfiguriert, dass er, wenn nicht anders erwähnt, 800 kW liefert. SUN kann ähnlich wie WIND Abweichungen aufweisen und hat als Grundproduktion 800 kW. Außerdem liefert FUEL, wie prognostiziert, 500 kW. Einer der Verbraucher wird in einigen Testfällen Abweichungen von Prognose und realem Verlauf aufweisen (siehe dazu Konfigurationen der Testfälle). Insgesamt werden also 2800 kW angeboten und benötigt.

12.2.1. Testfall 1a

Erwartungen

In diesem Testlauf ist die Prognosekorrektur ausgeschaltet. Die Agenten haben keine Möglichkeit der Nachverhandlung, d.h. jegliche Abweichung muss über den BRE abgerechnet werden, da auf dem Markt die angebotene Energie von den Verbrauchern bereits aufgekauft sein wird (siehe Konfiguration). WIND2 und SUN werden also Abrechnungen mit dem BRE durchführen, wobei WIND2 aufgrund der positiven Energieabwei-

chung auch einen positiven Saldo mit dem BRE aufweist, SUN mit seiner zu geringen Produktion einen negativen Saldo.

Konfiguration

Die Konfiguration erfolgt wie oben beschrieben. Der reale Verlauf weicht von der Prognose beim Agenten WIND2 von 00:30 bis 01:00 Uhr um +50 kW, von 01:30 bis 02:00 Uhr um +200 kW ab. Der restliche reale Verlauf gleicht der Prognose von 800 kW. Der Agent SUN ist passend zum Agenten WIND2 so konfiguriert, dass sein realer Verlauf von der durchgängigen Prognose (800 kW) von 00:30 bis 01:00 Uhr um -50 kW, von 01:30 bis 02:00 Uhr um -200 kW abweicht. Die Abweichungen der Agenten heben sich also auf.

Ergebnisse

Die Ergebnisse in diesem Testlauf folgten komplett den Erwartungen. Nach einer kurzen Zeit zur Initialisierung von ca. fünf Minuten haben beide Erzeuger die prognostizierte Leistung angeboten und anschließend verkauft. Die Abweichungen von der Prognose beeinflussen das Angebot nicht, das heißt, dass zuviel erzeugte Leistung an den BRE verkauft wird und zu wenig produzierte Leistung vom BRE bezogen wird.

12.2.2. Testfall 1b

Erwartungen

In diesem Testlauf ist die Prognosekorrektur deaktiviert, d. h. jegliche Abweichung der Prognose vom realen Verlauf muss über den BRE ausgeglichen und abgerechnet werden. Da in diesem Testfall der Produzent WIND2 und ein Verbraucher entsprechende Abweichungen aufweisen, werden diese beiden Agenten ihre Leistungsdifferenzen mit dem BRE abrechnen. Dabei gilt für sie, dass die Gesamtabrechnung einen Wert von 0 ergibt, da ihre Profile so konfiguriert sind, dass die negativen und positiven Differenzen sich mengenmäßig genau aufheben. Da der BRE hier identische Ein- und Zukaufspreise hat, können beide Agenten ihre Abrechnung ausgleichen. Bei den übrigen Agenten ist eine Abrechnung mit dem BRE nicht notwendig.

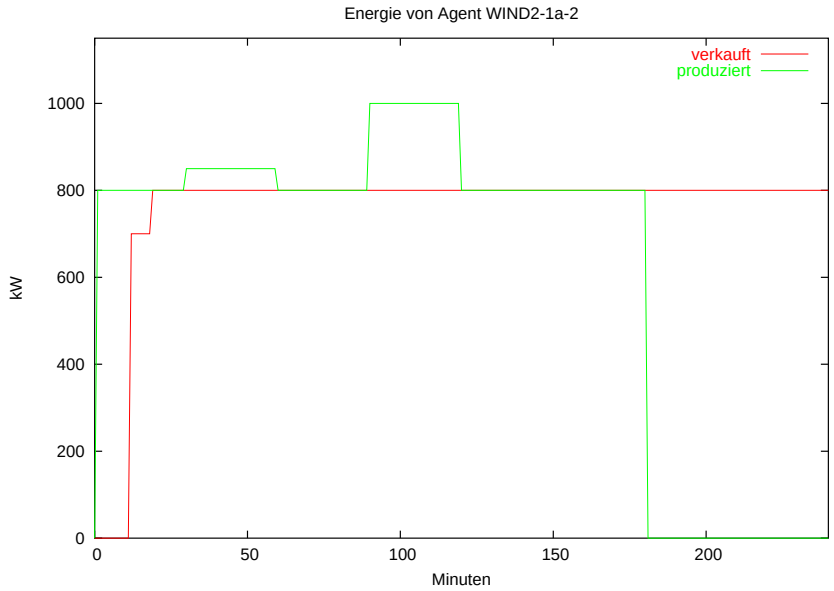

Abbildung 12.8.: Der Energieumsatz von PWind2-2 (Sz. 2 Fall 1a)

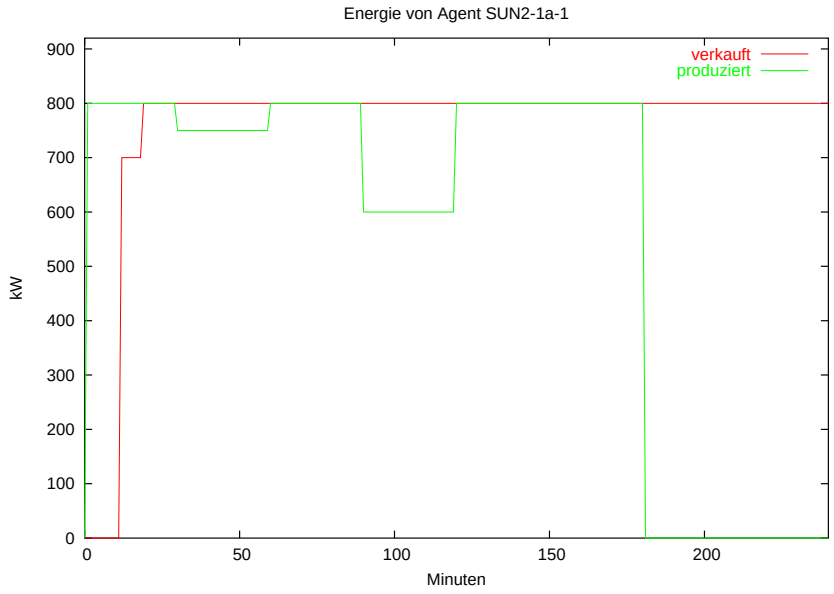

Abbildung 12.9.: Der Energieumsatz von PSun2 (Sz. 2 Fall 1a)

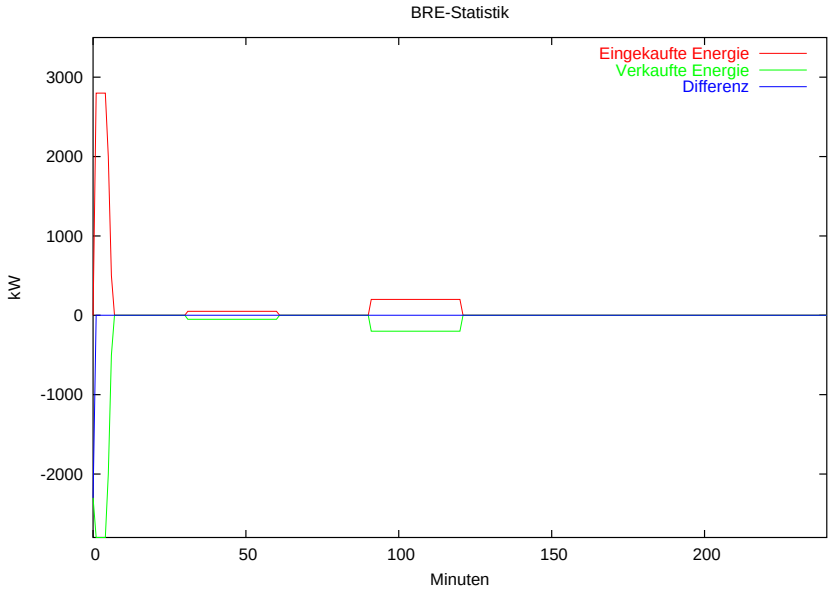

Abbildung 12.10.: Die Bilanz (Sz. 2 Fall 1a)

Bei ihnen stimmen Prognose und realer Verlauf überein. Desweiteren werden keine Verhandlungen zwischen dem Verbraucher und WIND2 stattfinden, um die Abweichungen auszugleichen.

Konfiguration

Angelehnt an die Grundkonfiguration weisen drei Verbraucher (Verbrauch je 700 kW), WIND1 (Produktion 700 kW), FUEL (Produktion 500 kW) und SUN (Produktion 800 kW) keine Abweichungen zwischen Prognose und realem Verlauf auf. Nur WIND2 und ein Verbraucher haben teilweise abweichende Prognosen. WIND2 (bzw. der Verbraucher) produziert (verbraucht) von 00:30 bis 01:00 Uhr unit[50]kW zuviel, von 01:30 bis 02:00 Uhr 200 kW zuviel, von 03:00 bis 03:30 Uhr 50 kW zu wenig und von 04:00 bis 04:30 Uhr 200 kW zu wenig. Diese Abweichungen der Agenten heben sich also auf.

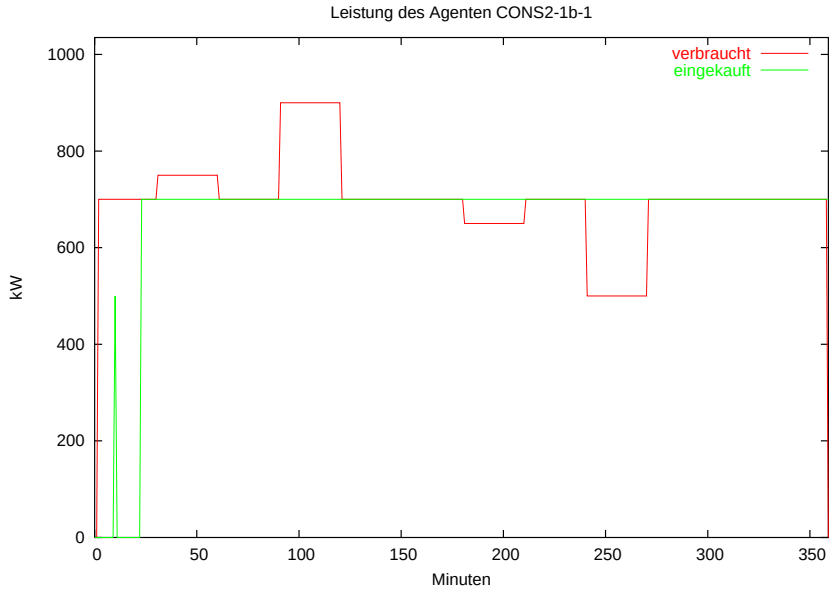

Abbildung 12.11.: Der Energieumsatz von CONS1 (Sz. 2 Fall 1b)

Ergebnisse

Die Ergebnisse in diesem Testlauf folgen komplett den Erwartungen. Nach einer kurzen Zeit zur Initialisierung haben sowohl der Verbraucher (siehe Abbildung 12.11) als auch der Erzeuger (siehe Abbildung 12.12) die prognostizierte Leistung nachgefragt und gekauft beziehungsweise angeboten und anschließend verkauft. Die Abweichungen von der Prognose beeinflussen das Angebot nicht, das heißt, dass zu wenig verbrauchte bzw. zu viel erzeugte Leistung an den BRE verkauft wird und zu viel verbrauchte bzw. zu wenig produzierte Leistung vom BRE bezogen wird. Die Bilanz des BRE ist in Abbildung 12.13 dargestellt; sie ist ausgeglichen.

12.2.3. Testfall 2a

Erwartungen

Da dieser Testfall an den Testfall 1a angelehnt und die Prognosekorrektur diesmal aktiviert ist, besteht die Möglichkeit der Nachverhandlung für die Agenten. Aufgrund der Abweichungen des realen vom prognostizierten

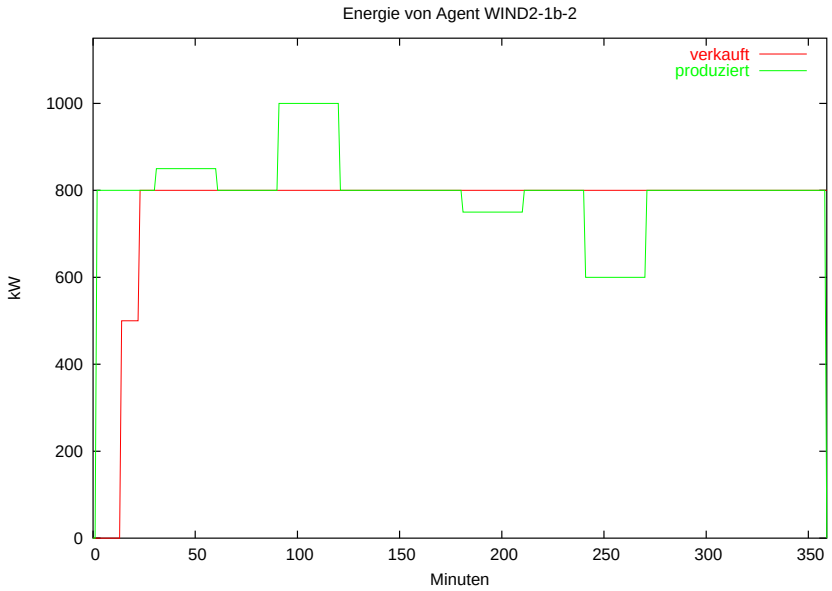

Abbildung 12.12.: Der Energieumsatz von PWind2 (Sz. 1 Fall 1b)

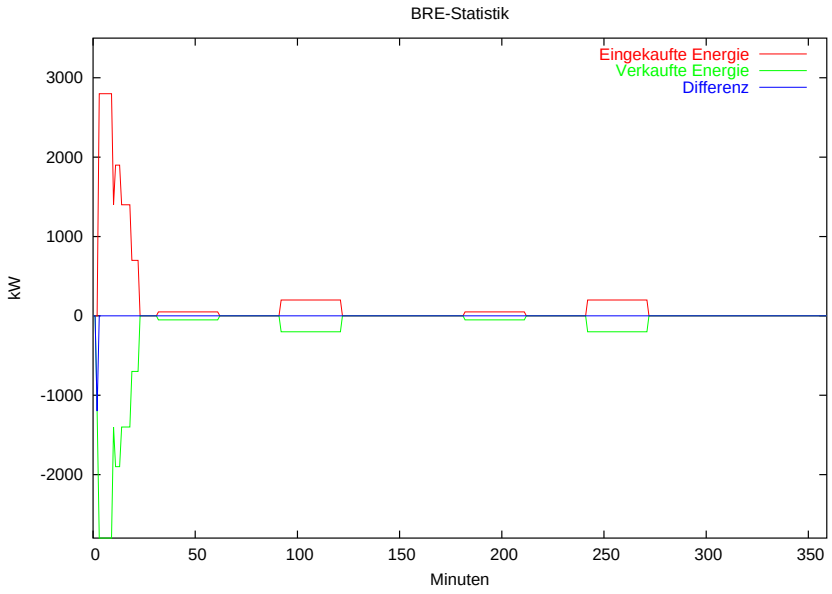

Abbildung 12.13.: Bilanz des BRE (Sz. 2 Fall 1b)

Verlauf bei den Agenten WIND2 und SUN sollte eine Nachverhandlung zwischen diesen beiden Agenten stattfinden. Dabei ist zu beachten, dass die Prognosekorrektur bei der jetzigen Strategie erst bei einer Abweichung von ±100 kW eingreift, d. h. die erste Abweichung von ±50 kW wird nicht korrigiert. Die abgerechneten Beträge der beiden Agenten mit dem BRE sollen im Vergleich zum Testfall 1a geringer ausfallen, da die Differenzen zum Teil untereinander ausgeglichen wurden. Sie werden aber nicht 0, da die Korrektur erst zum realen Verlauf stattfindet, also äußerst kurzfristig, und bei diesen sprunghaften Änderungen ein paar Minuten zum Ausgleich benötigt.

Konfiguration

Siehe Testfall 1a, nur mit eingeschalteter Prognosekorrektur und Möglichkeit der Nachverhandlung.

Ergebnisse

Die ersten Veränderungen in den realen Verläufen sind zu gering, um eine Prognosekorrektur zu bewirken, siehe Abbildungen 12.14 und 12.15. Erst die zweite Änderung ist groß genug, um eine Prognosekorrektur auszulösen.

Die Auswirkungen der Prognosekorrektur werden in den Abbildungen 12.16 und 12.17 deutlich. Nachdem sich der reale Verlauf verändert hat, passt sich die Korrektur zeitverzögert diesem an. Das zeigt, dass die Prognosekorrektur korrekt funktioniert. Das Ergebnis spiegelt sich auch in der BRE-Statistik wieder, die in Abbildung 12.18 angezeigt ist. Im Vergleich zu Testfall 1a fällt diese geringer aus, siehe Abbildung 12.10.

12.2.4. Testfall 2b

Erwartungen

Dieser Testfall hat die gleiche Konfiguration wie Testfall 1b, nur sind jetzt Prognosekorrektur und Nachverhandlung aktiviert. Es wird erwartet, dass nun der Verbraucher und WIND2 in Nachverhandlung treten, um ihre Defizite bzw. Überschüsse auszugleichen. Das bedeutet, dass die mit dem BRE verrechnete Leistung geringer wird. Das System soll sich zum Teil selbst intern ausgleichen. Die unterschiedlich starken Abweichungen (±50 kW,

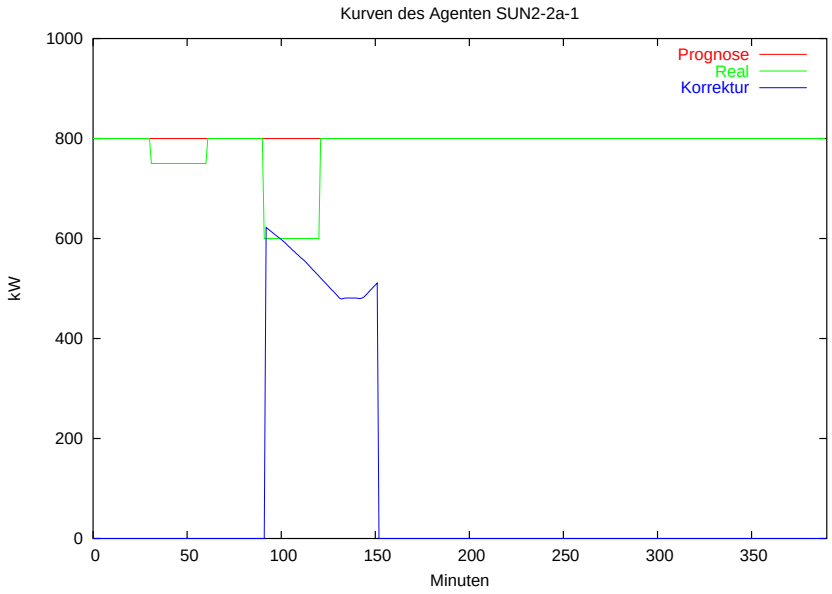

Abbildung 12.14.: Prognose und Prognosekorrektur von Sun (Sz. 2 Fall 2a)

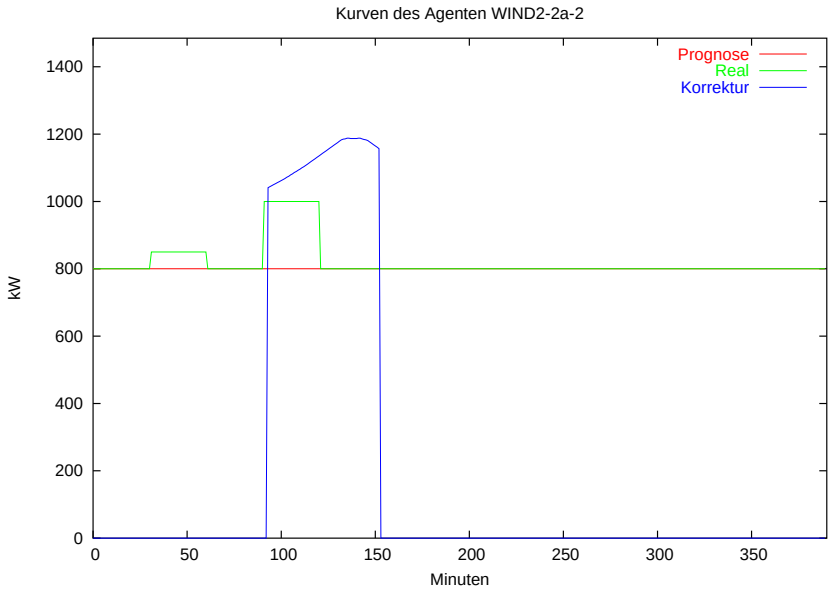

Abbildung 12.15.: Prognose und Prognosekorrektur von Wind (Sz. 2 Fall 2a)

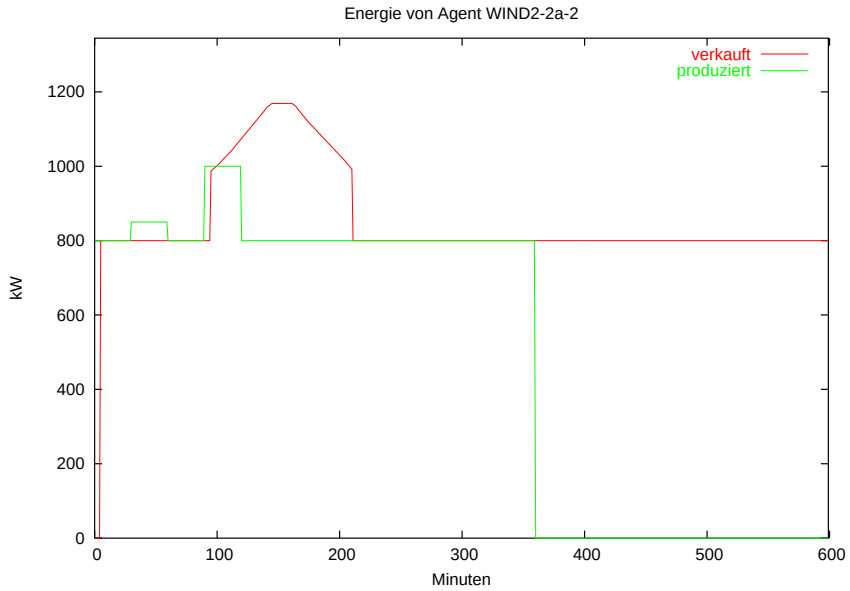

Abbildung 12.16.: Der Energieumsatz von PWind2 (Sz. 2 Fall 2a)

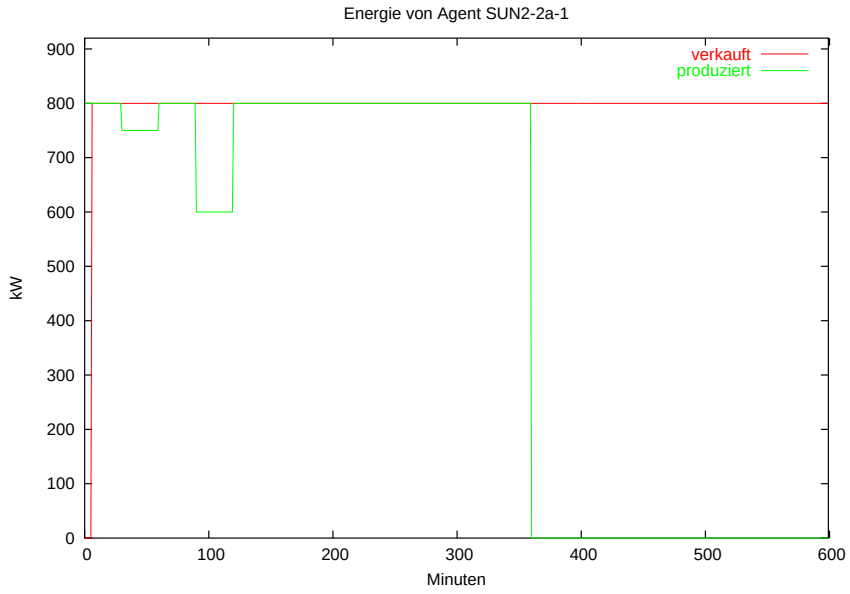

Abbildung 12.17.: Der Energieumsatz von PSun1 (Sz. 2 Fall 2a)

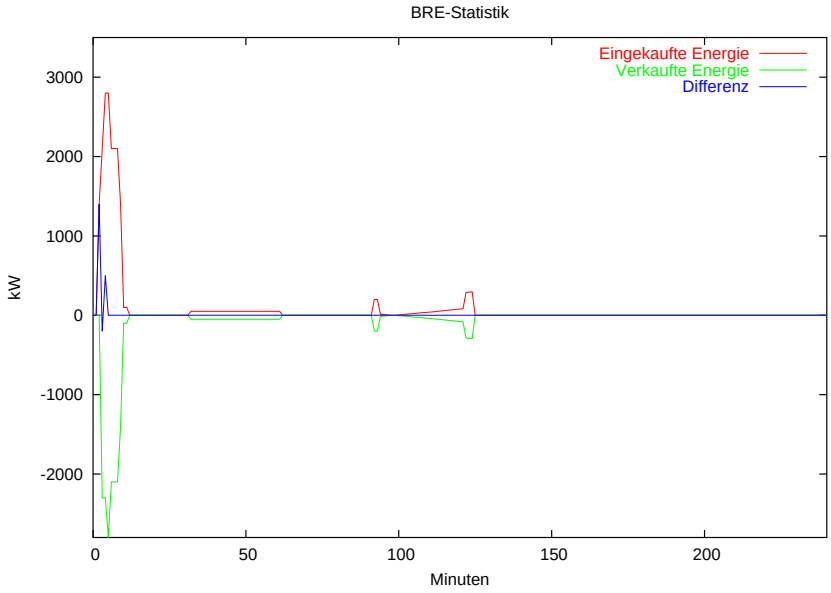

Abbildung 12.18.: Die Bilanz (Sz. 2 Fall 2a)

±200 kW) sollen zeigen, dass die Prognosekorrektur der Technik auf die sprunghaften Änderungen durch Nachverhandlungen gleichermaßen gut reagiert.

Konfiguration

Wie in Testfall 1b, nur mit eingeschalteter Prognosekorrektur und Möglichkeit der Nachverhandlung.

Ergebnisse

Auch hier ist zu sehen, dass die erste Veränderung im realen Verlauf keine Prognosekorrektur bewirkt. Erst die zweite Änderung ist groß genug, um eine Prognosekorrektur auszulösen. Dies ist in den Abbildungen 12.19 und 12.20 dargestellt.

Die Auswirkungen der Prognosekorrektur werden in den Abbildungen 12.21 und 12.22 deutlich. Nachdem sich der reale Verlauf verändert hat, passt sich die Korrektur diesem zeitverzögert an. Das zeigt, dass die Pro-

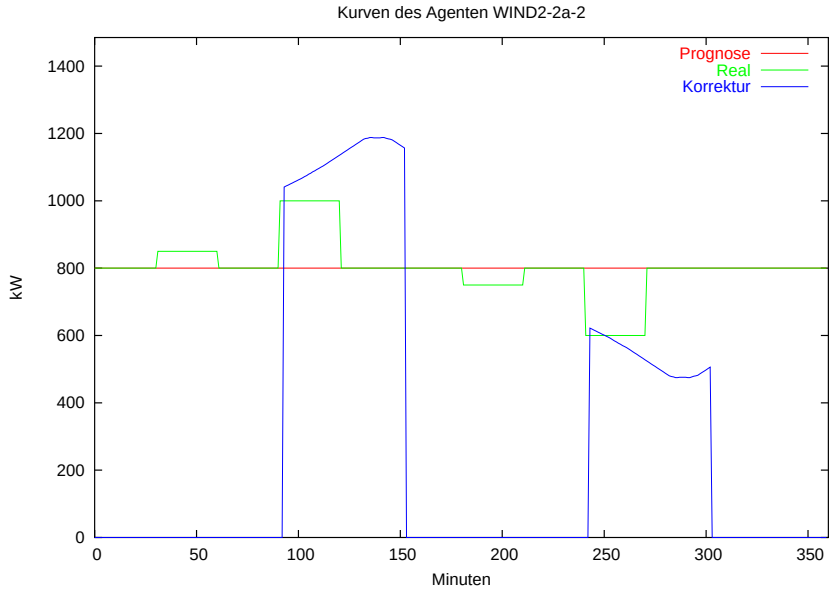

Abbildung 12.19.: Angebotsprognose, effektiv produzierte Energie und Prognosekorrektur des Erzeugers (Sz. 2 Fall 2b)

gnosekorrektur korrekt funktioniert. Das Ergebnis spiegelt sich auch in der BRE-Statistik wieder, die in Abbildung 12.23 angezeigt ist.

12.2.5. Testfall 3

Erwartungen

Im dritten Testfall sind Prognosekorrektur und Nachverhandlung aktiviert. Zusätzlich zu den bereits in Testfall 2 getesteten Funktionen wird hier getestet, ob die Agenten auf den Ausfall von Leitungssegmenten angemessen reagieren. Die Ausfälle von WIND2 bzw. eines Verbrauchers sollen von einer Prognoseänderung von SUN teilweise abgefangen werden. Diese Prognoseänderung ist fest vorgegeben. So sollen diese Ausfälle von SUN teilweise kompensiert werden und jeweils nicht die volle Leistungsdifferenz vom Bilanzkreis bezogen bzw. verkauft werden.

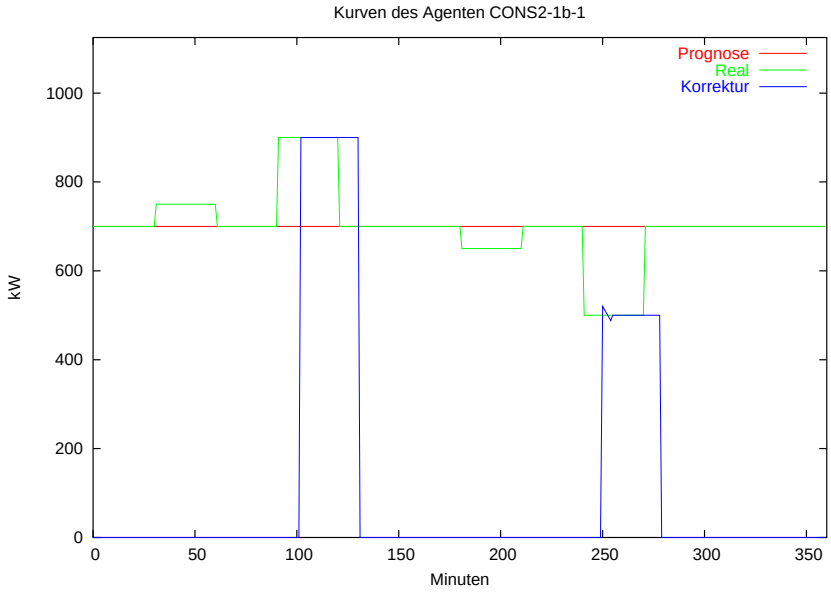

Abbildung 12.20.: Nachfrageprognose, effektiv nachgefragte Energie und Prognosekorrektur des Verbrauchers (Sz. 2 Fall 2b)

Konfiguration

Die zwei der drei Verbraucher (Verbrauch je 700 kW), WIND1 (Produktion 700 kW) und FUEL (Produktion 500 kW) weisen keine Abweichungen zwischen Prognose und realem Verlauf auf. WIND2 (Produktion 700 kW), Sun (Produktion 800 kW) und ein Verbraucher (Verbrauch 700 kW) haben teilweise von den angegebenen Werten abweichende Prognosekorrekturen sowie Leitungsausfälle.

- WIND2 hat von 00:30 bis 01:00 Uhr einen Ausfall seines Leitungssegments und kann keine Leistung einspeisen. In diesem Zeitraum kann keiner der anderen Agenten den Ausfall ausgleichen.
- Von 02:00 bis 02:30 Uhr hat WIND2 erneut einen Leitungsausfall. In genau demselben Zeitraum ändert sich für SUN die Prognose, so dass SUN hier 400 kW mehr produzieren kann. Also sollte WIND2 durch Nachverhandlungen mit SUN einen Teil seines durch den Ausfall entstandenen Energiebedarfs decken können.

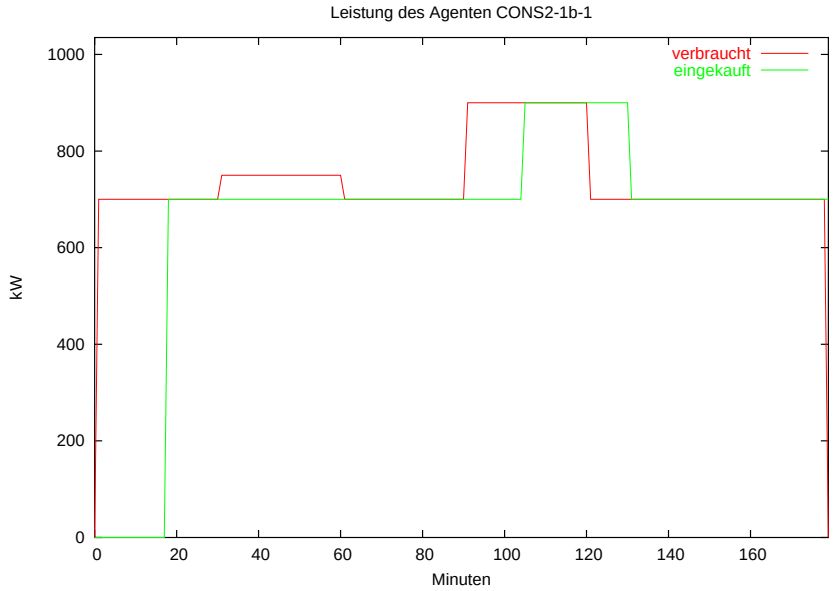

Abbildung 12.21.: Vom Verbraucher gekaufte und verbrauchte Leistung (Sz. 2 Fall 2b)

- Von 03:30 bis 04:00 Uhr hat WIND2 einen dritten Leitungsausfall. Hier ändert sich die Prognose eines Verbrauchers, so dass er 400 kW weniger verbraucht. Auch hier sollte WIND2 einen Teil seines entstandenen Strombedarfs durch Nachverhandlung decken können.

- Ein Verbraucher hat von 05:00 bis 05:30 Uhr einen Ausfall seines Leitungssegments und kann keine Leistung verbrauchen. Er wird die Leistung, die er für diesen Zeitraum bereits gekauft hat, wieder anbieten. Die anderen Agenten sind jedoch so konfiguriert, dass sie ihren Bedarf bereits gedeckt haben und der Verbraucher keinen Abnehmer finden wird.

- Von 06:30 bis 07:00 Uhr hat der Verbraucher erneut einen Ausfall seines Leitungssegments und wird die 700 kW, die er in diesem Zeitraum nicht verbrauchen kann, erneut anbieten. In diesem Zeitraum hat SUN eine Prognosekorrektur, so dass er 400 kW weniger als bisher vorgegeben produzieren wird. SUN und der Verbraucher mit dem

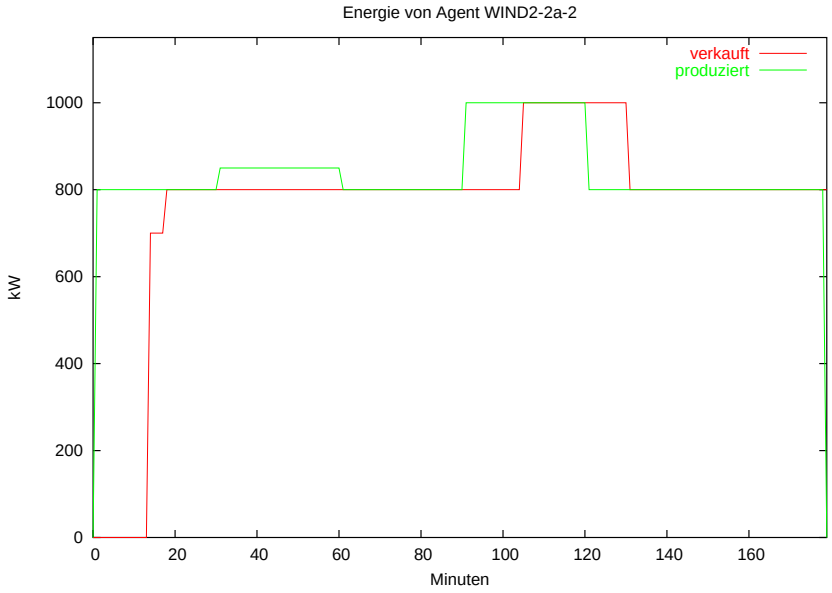

Abbildung 12.22.: Vom Erzeuger verkaufte und produzierte Leistung (Sz. 2 Fall 2b)

Leitungsausfall sollten in diesem Zeitraum in Nachverhandlung treten, so dass beide einen Teil ihrer Schwankungen ausgleichen können.

Ergebnisse

Die Leitungssegmentausfälle von WIND2 sind, siehe Abbildung 12.27, wie erwartet eingetreten. Der erste Ausfall konnte nicht kompensiert werden. Dies ist in der Bilanz des BRE zu sehen, siehe Abbildung 12.24, und entspricht unseren Erwartungen. Der zweite Ausfall von WIND2 wurde erwartungsgemäß teilweise durch SUN kompensiert, siehe Abbildung 12.26 und 12.24. Da die Kompensierung nur durch eine Prognosekorrektur stattfinden kann, sind die Werte im BRE nicht exakt. Die Prognosekorrektur berechnet wahrscheinliche, neue Werte und kennt nicht die genaue Zukunft. Dadurch kommen die etwas ungeordnet wirkenden Kurven für die verkaufte bzw. gekaufte Energie des BRE zustande. Der dritte Ausfall von WIND3 wurde erwartungsgemäß teilweise von dem Verbraucher kompensiert, siehe Abbildung 12.25.

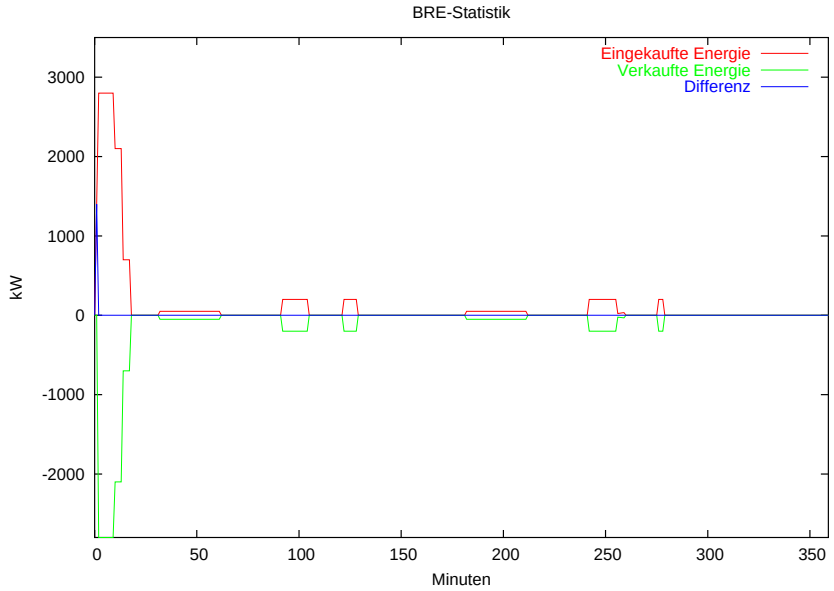

Abbildung 12.23.: Leistung, die an den BRE verkauft bzw. vom BRE gekauft wurde (Sz. 2 Fall 2b)

Der erste Leitungssegmentausfall des Verbrauchers ist von Minute 300 bis 330 eingetreten. Wie erwartet hat hier keine Kompensation von einem anderen Agenten stattgefunden. Der zweite Leitungssegmentausfall des Verbrauchers von Minute 390 bis 420 wird teilweise von SUN ausgeglichen, da SUN hier Energie von dem Verbraucher einkauft, siehe Abbildung 12.26. Die Ergebnisse des Tests entsprechen also unseren Erwartungen.

12.3. Testszenario Drei

Nachdem im zweiten Testszenario die grundlegenden Fähigkeiten zur Prognosekorrektur und Nachverhandlung getestet wurden sowie die angemessene Reaktion auf den Ausfall von Leitungssegmenten, werden in diesem Szenario die verschiedenen Strategien der Erzeuger-Agenten getestet. Es soll herausgefunden werden, unter welchen Bedingungen welche Strategie das optimale Ergebnis liefert. Die einzelnen Testfälle werden für jeden Agenten Prognosekorrekturen sowohl nach oben als auch nach unten beinhalten; die reale Leistung wird also entweder niedriger, gleich oder höher

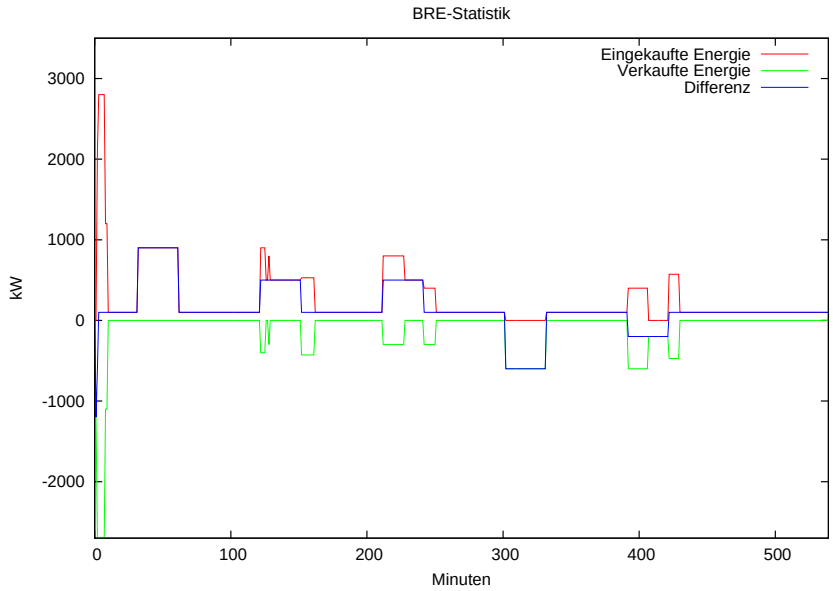

Abbildung 12.24.: „BRE“ (Sz. 2 Fall 3)

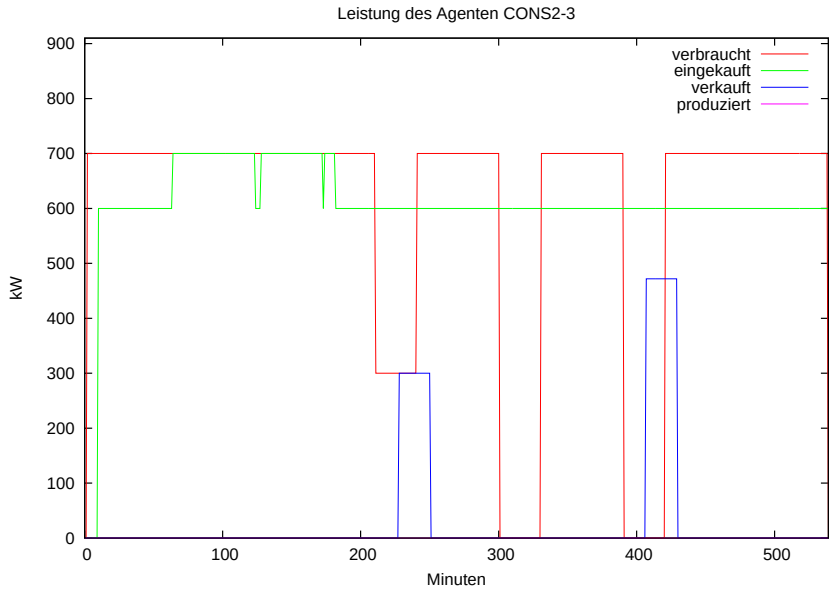

Abbildung 12.25.: „Verbraucher mit zwei Ausfällen.“ (Sz. 2 Fall 3)

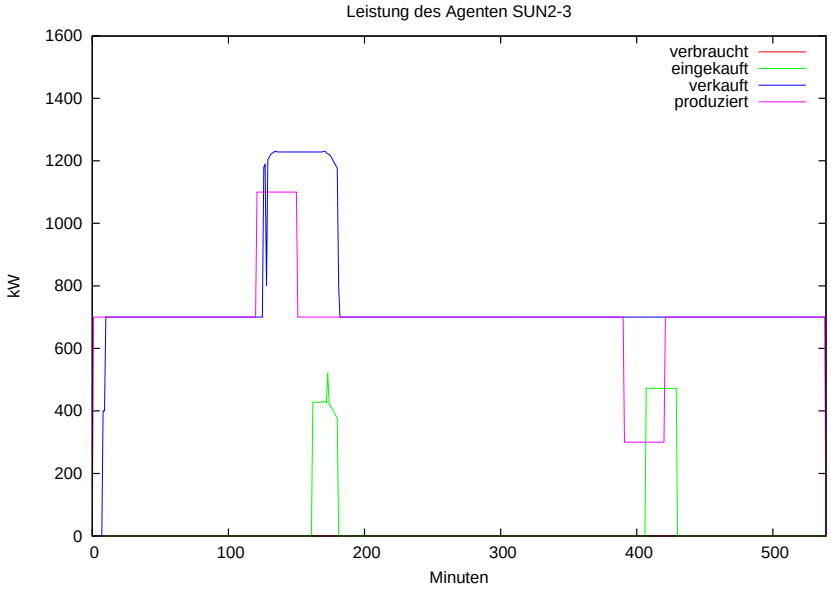

Abbildung 12.26.: „SUN mit Prognoseänderung und Produktionsanstieg.“ (Sz. 2 Fall 3)

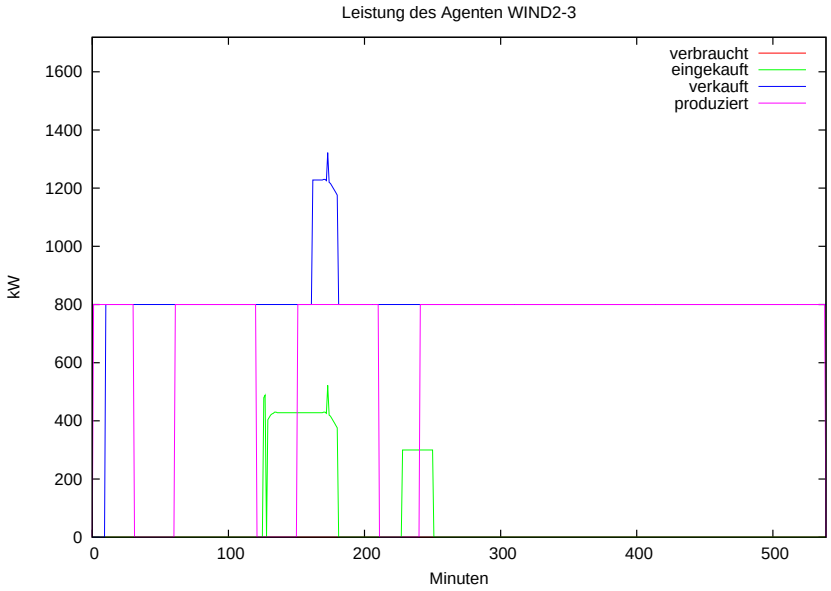

Abbildung 12.27.: „WIND2 mit drei Ausfällen.“ (Sz. 2 Fall 3)

als die prognostizierte sein. Die einzelnen Strategien sollten sich in den verschiedenen Situationen unterschiedlich verhalten.

Ein Beispiel zu den Erwartungen der Ergebnisse der Strategien des Erzeugers: Es gibt für den Erzeuger drei Strategien, eine risikolose, eine optimale (durchschnittliche) und eine riskante Variante. Die risikolose Variante bietet etwas weniger an, während der riskante Erzeuger ein ausgedehnteres Angebot wählt. Folglich ist zu erwarten, dass der risikolose Verbraucher bei einer Prognosekorrektur nach unten besser abschneidet als der riskante. Bei einer Prognosekorrektur nach oben sollte es genau umgekehrt sein.

12.3.1. Testfall 1

Der erste Testfall soll dazu dienen, herauszufinden, wie die Erzeuger mit unterschiedlichen Strategien auf die unterschiedlichen Änderungen der Prognose reagieren. Die Abweichungen des realen Verlaufs von der Prognose differieren untereinander mengenmäßig (Höhe der Abweichung) und zeitlich (sowohl kurze als auch länger andauernde Abweichungen). Außerdem ist die Geschwindigkeit der Änderung, gemessen in Differenz kW pro Minute, bei den verschiedenen Schwankungen unterschiedlich.

Erwartungen

Es ist zu erwarten, dass die unterschiedlichen Strategien der Agenten einen Einfluß auf die Energie haben, die sie anbieten und aufgrund der vorherrschenden Übernachfrage auch verkaufen. Schließlich reagieren die verschieden konfigurierten Agenten überhaupt erst bei unterschiedlichen Differenzen zur Prognose. Des weiteren erfolgt bei jeder der Strategien die Anpassung des Angebotes an die von der ursprünglichen Prognose abweichenden realen Werte mit einer unterschiedlichen Geschwindigkeit.

Konfiguration

In allen Varianten des ersten Testfalls ist ein einfaches Szenario vorgegeben. Es gibt nur einen Verbraucher mit einer konstanten prognostizierten und realen Nachfrage von 700 kW und einen Erzeuger WIND mit einer Prognose von konstant 50 kW. Nun gibt es folgende reale Verläufe, die von

der Prognose abweichen und somit eine Prognosekorrektur bewirken (siehe auch Abbildung 12.28).

- 00:00 - 00:30: Produktion von 50 kW
- 00:30 - 00:40: linearer Abfall der Leistung auf 0 kW
- 00:40 - 01:10: Produktion von 0 kW
- 01:10 - 01:20: linearer Anstieg der Leistung auf 50 kW
- 01:20 - 02:00: Produktion von 50 kW
- 02:00 - 02:30: linearer Anstieg der Leistung auf 250 kW
- 02:30 - 03:00: Produktion von 250 kW
- 03:00 - 03:30: linearer Abfall der Leistung auf 50 kW
- 03:30 - 04:10: Produktion von 50 kW
- 04:10 - 05:50: linearer Anstieg der Leistung auf 250 kW
- 05:50 - 07:30: Produktion von 250 kW
- 07:30 - 09:10: linearer Abfall der Leistung auf 50 kW
- 09:10 - 10:10: Produktion von 50 kW
- 10:10 - 10:30: linearer Anstieg der Leistung auf 100 kW
- 10:30 - 10:50: Produktion von 100 kW
- 10:50 - 11:10: linearer Anstieg der Leistung auf 200 kW
- 11:10 - 11:30: Produktion von 200 kW
- 11:30 - 11:50: linearer Abfall der Leistung auf 50 kW
- ab 11:50: Produktion von 50 kW

Wie leicht zu erkennen ist, haben die einzelnen Abweichungen von der Prognose unterschiedliche Verläufe. Die erste Abweichung liegt betragsmäßig bei 50 kW, die Leistung ändert sich in 30 Minuten um 50 kW. Die zweite Abweichung beträgt 200 kW, die Leistung ändert sich schneller, und zwar in 30 Minuten um 200 kW, usw.

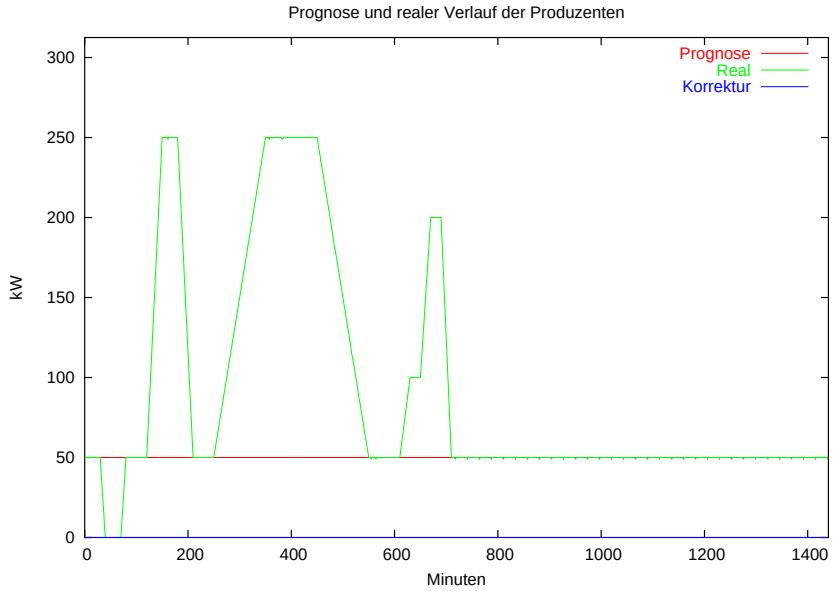

Abbildung 12.28.: Verlauf der Prognose und der realen Produktion (Sz. 3 Fall 1)

Ergebnisse des Testfalles 1a – „Medium“

Im Testfall 1a wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie *medium* initialisiert. Bei der ersten starken positiven Prognoseabweichung wird erheblich mehr Energie verkauft, bei der zweiten nicht so starken Abweichung wird nicht ganz so viel Energie verkauft. Abbildung 12.29 zeigt das Ergebnis.

Ergebnisse des Testfalles 1b – „None“

Im Testfall 1b wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie *none* initialisiert. Das führte zu folgenden Ergebnissen:

Wie erwartet zeigt Abb. 12.30, daß keine Prognosekorrektur stattfindet. Es wird genau 50 kW verkauft.

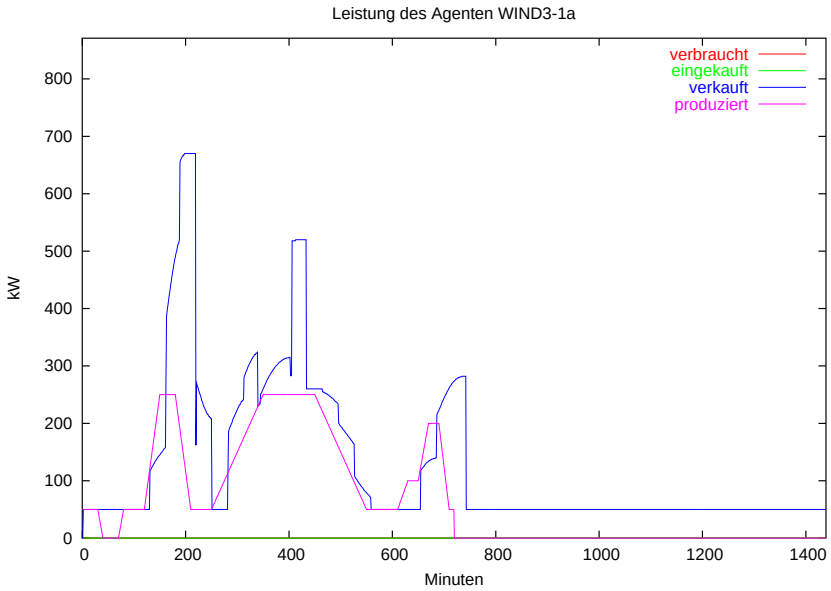

Abbildung 12.29.: Strategie *medium* (Sz. 3 Fall 1a)

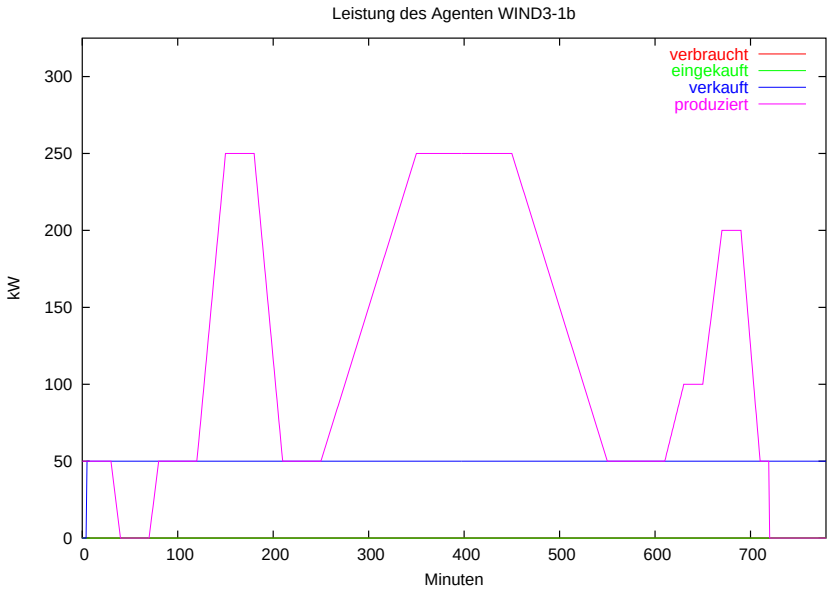

Abbildung 12.30.: Strategie *none* (Sz. 3 Fall 1b)

Ergebnisse des Testfalles 1c – „Dangerous“

Im Testfall 1c wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie „Dangerous“ initialisiert. Im Diagramm 12.31 ist zu sehen, dass die positiven Prognoseänderungen sofort bewirken, dass mehr Energie verkauft wird. Auch bei Rückgang der Produktion geht die Kurve der verkauften Energie erst langsam zurück.

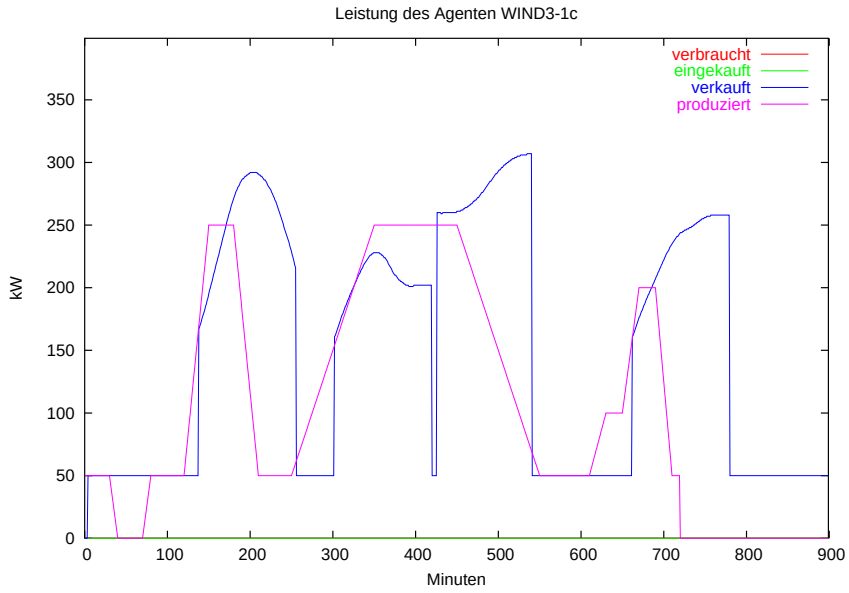

Abbildung 12.31.: Strategie *dangerous* (Sz. 3 Fall 1c)

Ergebnisse des Testfalles 1d – „No Risk“

Im Testfall 1d wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie *no risk* initialisiert. Es ist klar zu sehen, dass sich der Verlauf der verkauften Energie stark an den Verlauf der produzierten Energie anpasst. Die Ergebnisse sind in den Abbildungen 12.32 zu sehen.

Ergebnisse des Testfalles 1e – „Linear Short“

Im Testfall 1e wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie „Linear Short“ initialisiert. Das führte zu den in den Abbildun-

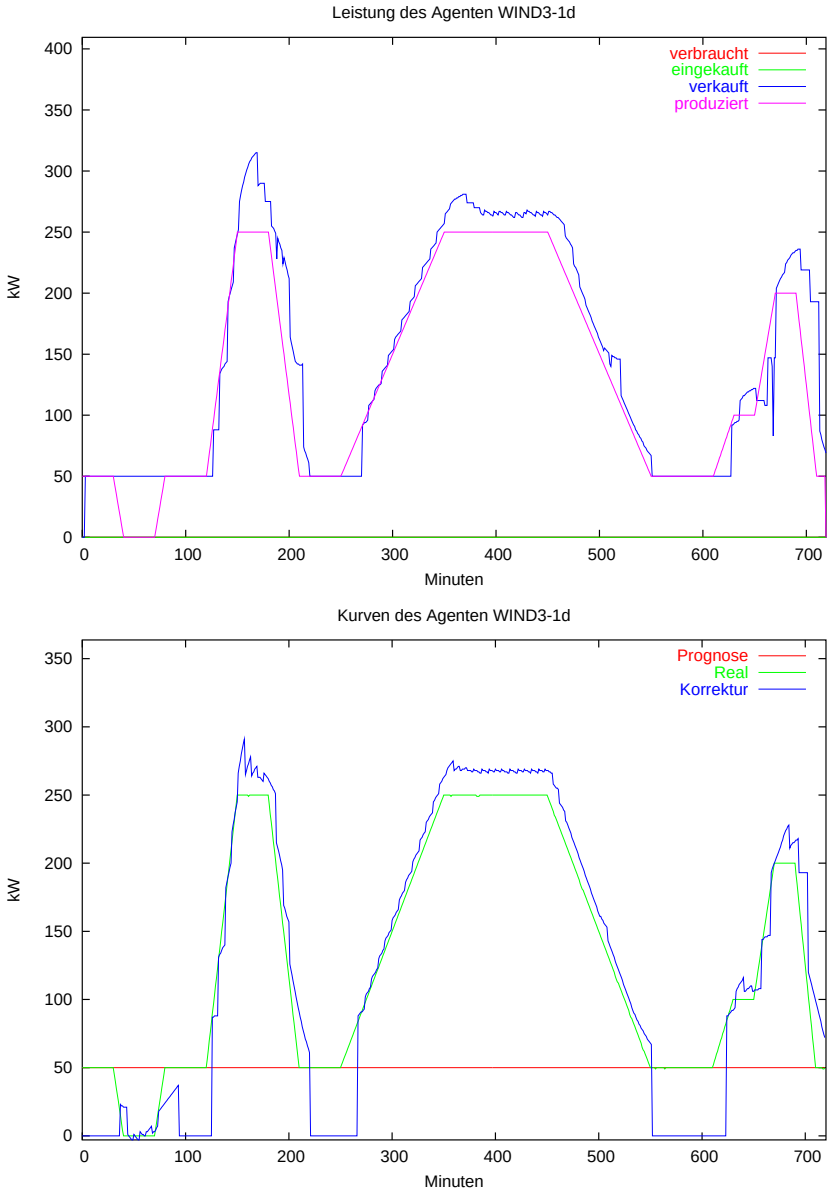

Abbildung 12.32.: Strategie *no risk* (Sz. 3 Fall 1 d)

gen 12.33 zu sehenden Ergebnissen. Hier passt sich der Verkauf schnell der Produktion an.

Ergebnisse des Testfalles 1f – „Linear Long“

Im Testfall 1f wurde im oben beschriebenen Szenario der Erzeuger mit der Strategie „Linear Long“ initialisiert. Die Ergebnisse sind in den Abbildungen 12.34 dargestellt. Hier ist zu sehen, dass sich der Verkauf langsamer der Produktion anpasst, als bei der Strategie *linear short*.

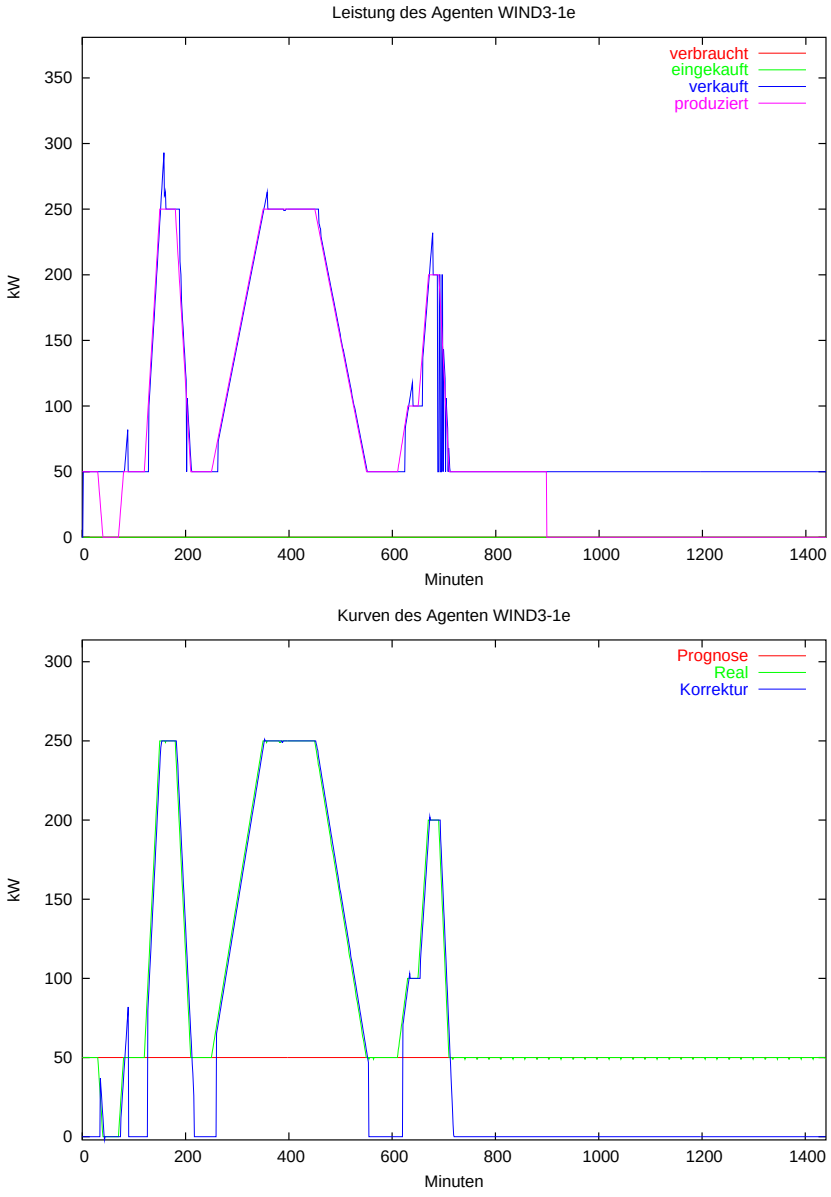

Abbildung 12.33.: Strategie *linear short* (Sz. 3 Fall 1e)

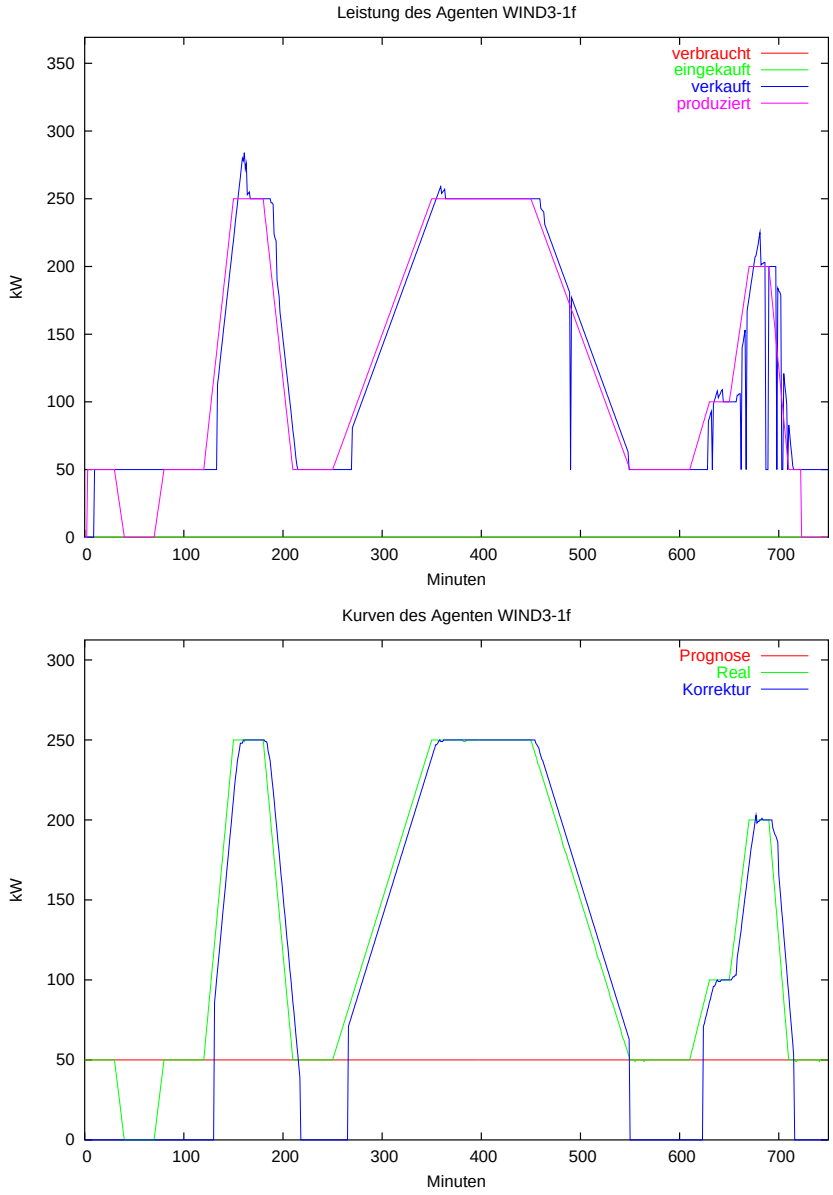

Abbildung 12.34.: Strategie *linear long* (Sz. 3 Fall 1f)

12.3.2. Testfall 2

Dieses Testszenario soll die Auswirkung der Strategien der Producer auf deren Verkaufsverhalten bei Überangebot testen. Es soll dabei festgestellt werden, welche Strategie am erfolgreichsten ist, also von welchem bzw. welchen Agenten die Energie eingekauft wird. Zu diesem Zweck hat das Testszenario die folgenden Teilnehmer:

- fünf Erzeuger, jeweils einen pro Strategie
- einen einzigen Verbraucher

Erwartungen

Der Erzeuger sollte seinen Bedarf zu Beginn der Simulation für den ganzen Tag decken können. Er sollte seine Energie von dem bzw. den Agenten beziehen, die an schnellsten Angebote am Blackboard einstellen können. Die Produzenten werden alle Angebote unterschiedlicher Länge am Blackboard einstellen. Je nach Geschwindigkeit der Angebotserstellung und der Einkaufsstrategie des Verbrauchers wird sich bei einigen Producern ein Vorteil bei der verkauften Energiemenge zeigen.

Konfiguration

Sowohl der Verbraucher als auch die Erzeuger haben eine Verbraucher-/Erzeugerprognose von 200 kW. Damit herrscht also ein Überangebot an Energie sodass der Verbraucher keine Probleme haben sollte, seinen Bedarf zu decken. Der reale Verlauf sieht folgendermaßen aus:

von [min]	bis [min]	Dauer [min]	Progn. [kWh]	Real [kWh]	Kommentar
0	60	60	200	200	Um das System anlaufen zu lassen.
60	120	60	200	200 → 300	schneller Anstieg mit 60° Steigung auf 300 kW
120	180	60	200	200	Um das System wieder zu stabilisieren.
180	353	173	200	200 → 300	langsamer Anstieg mit 30° Steigung auf 300 kW

Ergebnisse

Wie in dem Diagramm in Abbildung 12.35 zu erkennen ist, hat der Verbraucher wie erwartet keine Schwierigkeiten, seinen Bedarf zu decken. Auch die nicht vorhergesagten Änderungen werden durch die Prognosekorrektur erkannt und mit einer kurzen Verzögerung ausgeglichen.

Die Produzenten mit den Strategien „linear short“ und „linear long“ haben, wie in den Diagrammen in Abbildung 12.38 und 12.37 zu erkennen ist, nur einen geringen Teil ihrer Produktion verkaufen können.

Der größte Anteil der verbrauchten Energie wurde von den Agenten mit den Strategien *dangerous* und *no risk* gekauft (siehe Abbildungen 12.36 und 12.39). Ein erster großer Block wurde von vom Agenten mit der Strategie *dangerous* verkauft. Danach wurde die Energie abwechselnd von diesen beiden Agenten gekauft.

Die folgende Tabelle zeigt die verkaufte Energiemenge der einzelnen Agenten bei den Abweichungen von der Prognose:

Strategie	01:00:00–01:59:59 [kWmin]	03:00:00–05:52:59 [kWmin]
dangerous	767	2241
no risk	201	1755
medium	86	1175
linear long	76	145
linear short	211	161

12.4. Fazit aus den Testläufen

In den drei oben beschriebenen Testszenarien wurde das System und seine Funktionen schrittweise getestet.

12.4.1. Szenario Eins

Im ersten Szenario wurde mit einem einfachen Testlauf begonnen. Die Leistungsmengen und die Preise waren über den Testzeitraum konstant, und die Prognose entsprach den realen Werten. Bereits an dieser Stelle traten, trotz einer Testphase, die deutlich intensiver war als noch in dem ersten Semester von DEZENTEN, die ersten Probleme im Zusammenspiel der einzelnen Module auf. Diese konnten aber relativ zügig behoben werden. Bereits

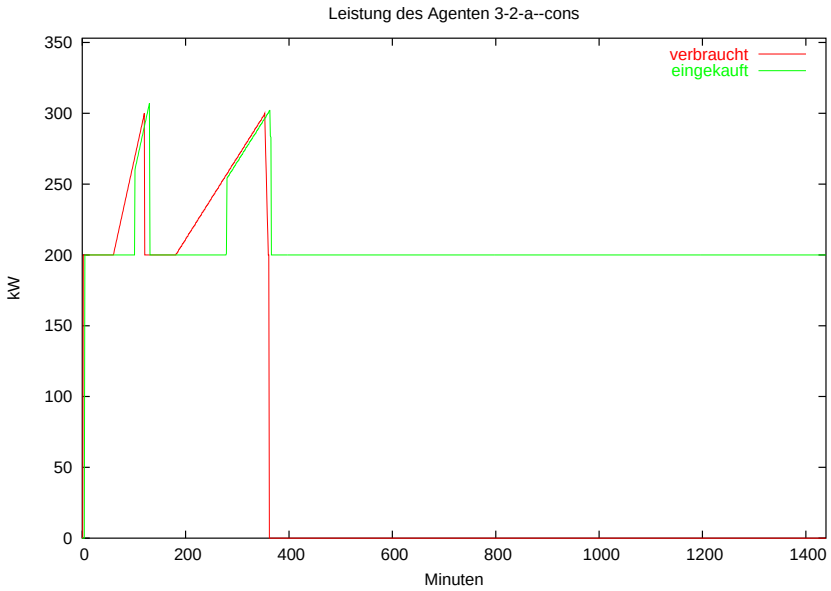

Abbildung 12.35.: Verbrauch (Sz. 3 Fall 2)

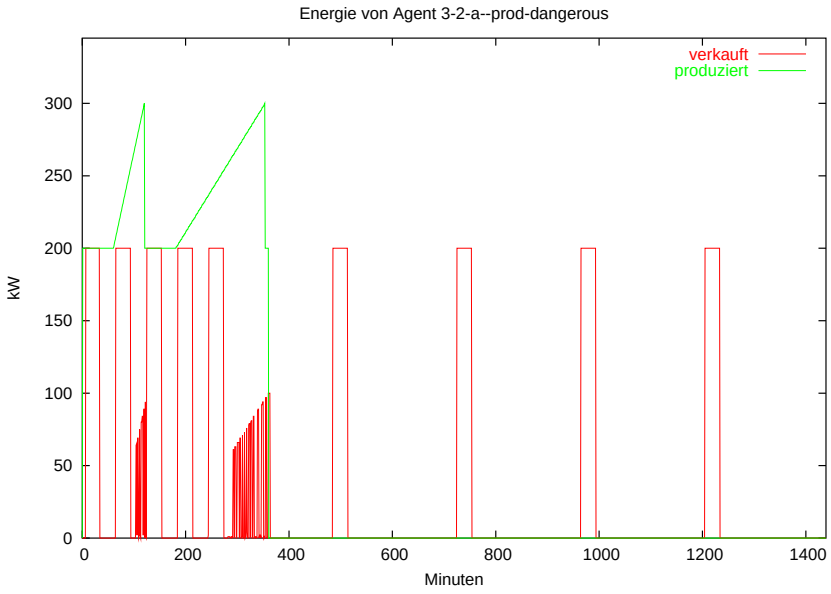

Abbildung 12.36.: Strategie „Dangerous“ (Sz. 3 Fall 2)

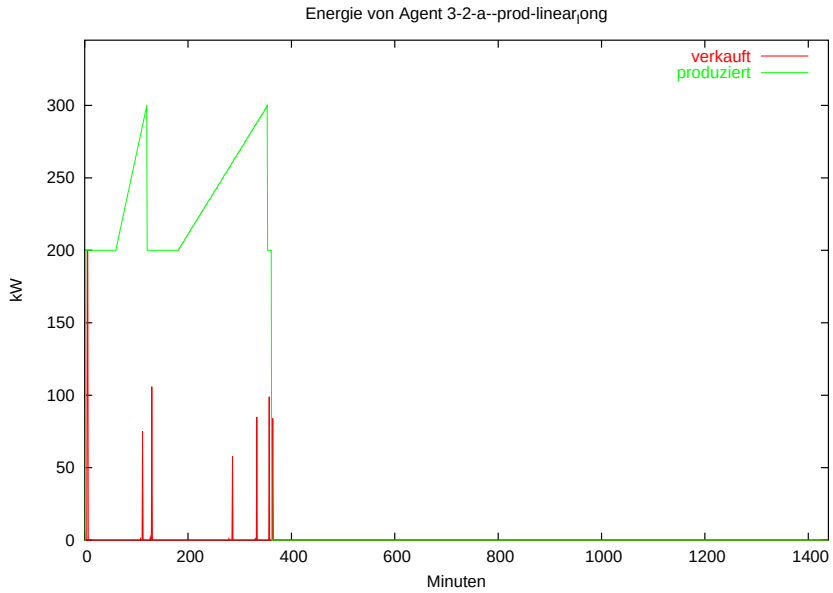

Abbildung 12.37.: Strategie „Linear Long“ (Sz. 3 Fall 2)

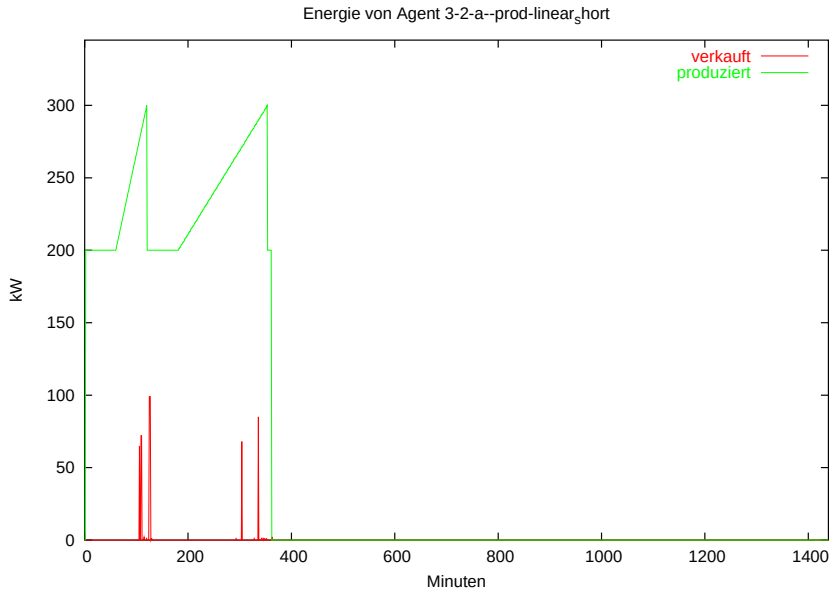

Abbildung 12.38.: Strategie „Linear Short“ (Sz. 3 Fall 2)

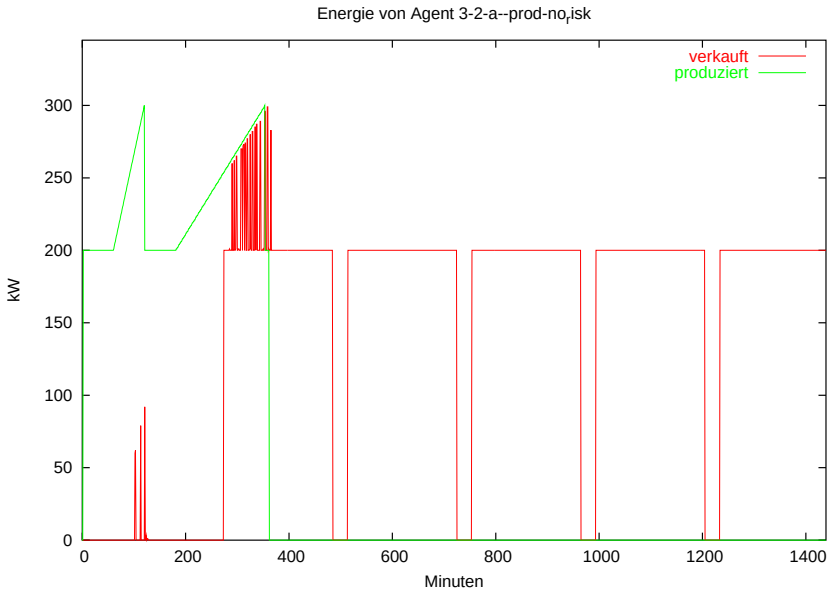

Abbildung 12.39.: Strategie „No Risk“ (Sz. 3 Fall 2)

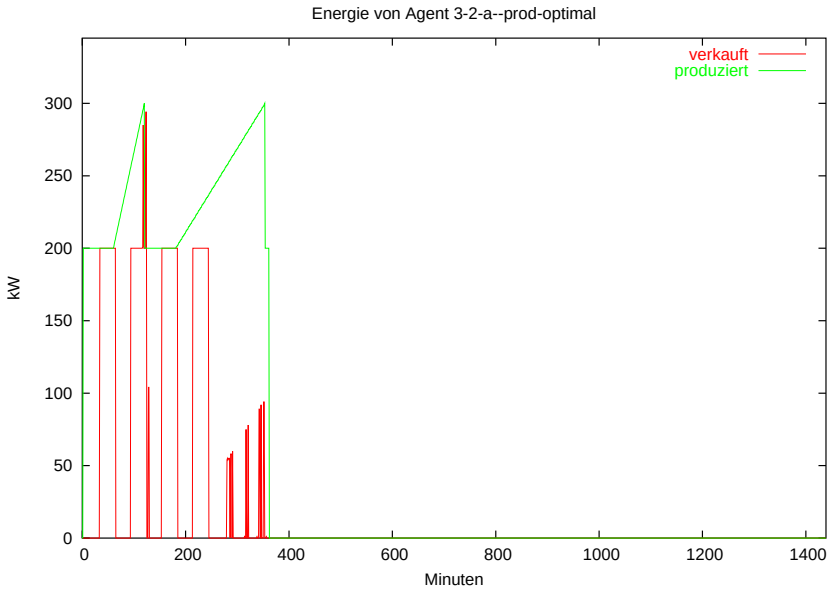

Abbildung 12.40.: Strategie „Medium“ (Sz. 3 Fall 2)

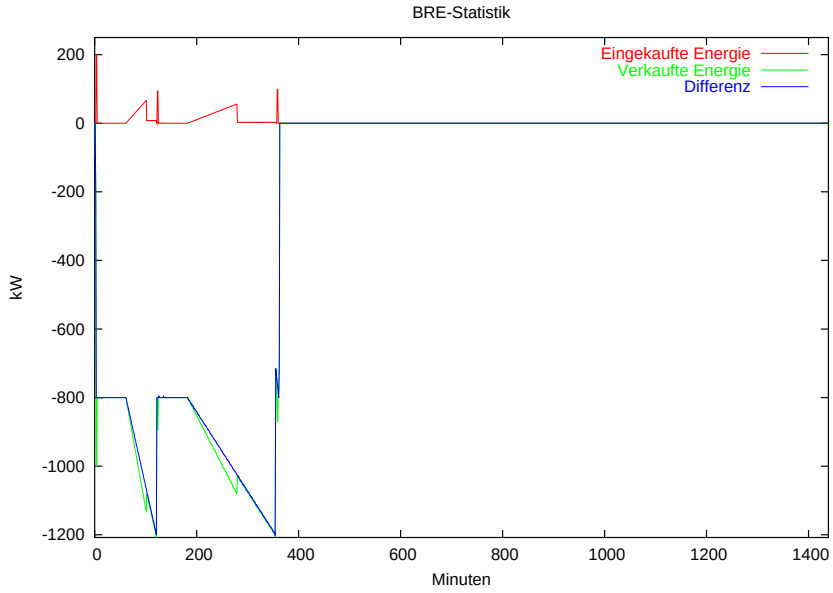

Abbildung 12.41.: BRE (Sz. 3 Fall 2)

hier erwies sich die Vorgehensweise, mit einer sehr einfachen Konfiguration anzufangen, und dafür eine höhere Zahl an Szenarien zu entwerfen, als die richtige Entscheidung.

12.4.2. Szenario Zwei

Nachdem die wenige Probleme mit der Prognosekorrektur beseitigt waren, und diese fehlerfrei lief, wurden die Prognosekorrektur, der Ausfall von Leitungssegmenten und die Nachverhandlungsmöglichkeit im Rahmen des zweiten Szenarios getestet. Es erwies sich als günstig, zuerst alle Komponenten einzeln zu testen, bevor in abschliessenden Testläufen auch das Zusammenspiel getestet wurde. Es wurde gezeigt, dass die Agenten angemessen auf Leitungsausfälle und Prognoseänderungen reagieren und auch die Nachverhandlung nutzen.

12.4.3. Szenario Drei

Nach dem zweiten Szenario traten keine weiteren entscheidenden Fehler auf. So wurde im dritten Szenario der von den Ergebnissen her interessanteste Testlauf gestartet. Hierbei wurden die Erzeuger- und Verbraucher-Agenten mit verschiedenen Strategien konfiguriert, um miteinander verglichen werden zu können. Die Effizienz der einzelnen Strategien wird deutlich, wenn die realen Werte deutlich von den Prognosewerten abweichen und die unvorhersehbare Produktion den Bedarf übersteigt.

Die linearen Prognosetrategien schnitten trotz der linear wachsenden Abweichungen für kurzfristige Nachverhandlungen durchweg am schlechtesten ab. Als am stärksten erwiesen sich „extremen“ Strategien „dangerous“ und „no risk“, wobei „dangerous“ konzipiert was, schnell relativ große Leistungen anzubieten; von „no risk“ war das Gegenteil erwartet worden. Interessant ist, dass die Einschätzung der Prognosestrategien mit gleitendem Durchschnitt zum Zeitpunkt der Planung, die sich in der Benennung zeigt, der tatsächlichen Performanz nicht entspricht.

Teil IV.

Schlussbemerkungen

13. Ergebnisse der Projektgruppe

Zusammenfassend lässt sich feststellen, dass die Projektgruppe alle *Ziele* erreicht hat. Die Konzentration – innerhalb des zunächst sehr weit gesteckten PG-Themas – auf die von der PG gesetzten *Schwerpunkte* (Verhandlungsmechanismen, Prognosestrategien) hat sich als sinnvoll erwiesen.

Die *Zeitplanung* des zweiten Semesters profitierte von den Erfahrungen des ersten. Dennoch war der Aufwand für die Integration der Module und für den Gesamttest größer als zuvor angenommen. Dennoch wurde die Projektgruppe termingerecht abgeschlossen. Der Termin für die Endpräsentation wurde auf den 30. April 2004 gelegt. Die positive Resonanz auf diese Präsentation ist bereits in Kapitel 1 beschrieben.

Bei der Entwicklung des implementierten Verhandlungsmechanismus galt es für alle Beteiligten, Neuland zu betreten. Bekannte Börsenmechanismen ließen sich nicht einfach übertragen. Die Projektgruppe konzentrierte sich daher zunächst auf einfache Verhandlungsmechanismen, die eine Bedarfsdeckung der beteiligten Akteure innerhalb technischer Grenzen sicher stellen. Dieser Anforderung entspricht das entwickelte Multiagentensystem. Detailierte Preisverhandlungen werden ein Gegenstand zukünftiger Arbeiten sein.

Die entwickelten *Prognoseverfahren* sind für die gestellten Anforderungen völlig ausreichend. Sie erreichen eine hohe Genauigkeit für kurzfristige Abweichungen, die z. B. durch unvorhersehbaren Bedarf entstehen. Die Ergebnisse sollen auf dem Workshop on Real-Time Programmung (WRTP'04) in Istanbul präsentiert werden.

Die im zweiten Semester entwickelte *graphische Benutzerschnittstelle* hat sich als zweckmäßig erwiesen und hat Konfiguration, Testen und Analyse in der zweiten Phase erheblich vereinfacht.

Noch zu lösende und weiterführende Probleme und Aufgabenstellungen sind im folgenden Kapitel aufgeführt.

14. Zukünftige Arbeiten

Als zukünftige Arbeiten einer Nachfolge-PG oder anschließender Diplomarbeiten werden folgende Verbesserungen vorgeschlagen:

Nachfragehandel Bisher werden nur Angebote gehandelt. Die Erzeuger erfahren nur indirekt (Preisentwicklung) den Bedarf der Verbraucher. Der Nachfragehandel ermöglicht den Verbrauchern, ihren Bedarf den Erzeugern mitzuteilen. Diese können dann ggf. ihre Produktion anpassen.

intelligentere Agentenstrategien Die Agenten sollen mit Hilfe der aktuellen Preisentwicklungen die Zukunft prognostizieren und entsprechend die Käufe und Verkäufe vorziehen bzw. verzögern.

zweistufige Verhandlung und techn. Randbedingungen Die Entscheidung, Energie zu produzieren oder abzunehmen ist nicht nur vom Preis sondern auch von technischen Bedingungen, Anlauf und Abschaltzeiten von Erzeugern und Verbrauchern, Kosten für Standby-Betrieb, etc., abhängig. Ein sinnvoller Einsatz setzt eine langfristige Planung voraus, die den von dieser Projektgruppe implementierten kurzfristigen Verhandlungen vorausgeht.

Reserve-Handel Für einen zuverlässigen Betrieb elektrischer Versorgungsnetze ist die Bereitstellung von Reserve-Leistung notwendig. Die Kosten für diese Dienstleistung müssen auf die anderen Akteure (abhängig von deren möglichen Abweichungen von den prognostizierten Leistungen) verteilt werden.

Umsetzung auf RT-Java Alle Deadlines des Systems sind firm Deadlines in einem Minutenraster. Java reicht für diese Anforderungen aus. Bei größeren Multiagenten-Systemen und Hierarchien wird eine Umstellung auf eine Programmiersprache nötig, die Realzeit-Bedingungen unterstützt.

Skalierbarkeit Erweiterung des Bilanzkreises auf mehrere Verhandlungsknoten und Zusammenarbeit mehrerer Bilanzkreise über ein Verbundnetz.

Simulation elektrischer Parameter Die Netzfrequenz f, die Spannung U und der Phasenwinkel ϕ geben jedem Akteur in Echtzeit Aufschluss über den Zustand des elektrischen Netzes. Diese Parameter sollen den Akteuren als Simulation zur Verfügung stehen.

Teil V.

Anhang

A. Abkürzungen

BB Blackboard

BKV Bilanzkreisverantwortlicher

BRE BalanceResponsibleEntity

DUI Dezenten User Interface

VA Verbraucheragent

VN Verbundnetz

B. Pakete und Klassen

B.1. Agent

- Agent
- AgentDispatcherTarget
- AgentThread
- PTest
- threadtest

B.2. BB

- AgentWrapper
- BlackBoard
- BlackboardDispatcherTarget
- NotifyAgentsThread
- Statistics

B.3. BRE

- BalanceResponsibleEntity
- BREDispatcherTarget

B.4. DezentenLib

- AgentType
- ClientID
- DataSourceType
- DDate
- DezException
- Filter
- ID
- Interval
- Offer
- OfferEventType
- OfferList
- OfferPriceComparator
- OfferRankComparator
- OfferTimeComparator
- PowerEventType
- Table

B.5. DezentenLib.DataSource

- ConstDataSource
- DataSource
- IntervalDataSource

- ModifiableDataSource
- NoDataException
- NoFutureException
- Point
- Pointlist
- PointXComparator
- SimpleInterval
- Spline

B.6. DezentenLib.Event

- AgentEvent
- BBEvent
- BREEvent
- Event

B.7. DezentenLib.Net

- AgentReceiver
- BlackboardReceiver
- BREReceiver
- ControllerReceiver
- DispatcherTarget
- Message
- NetAddress

- Network
- Receiver

B.8. Simulation

- AgentConfigListener
- AgentEventListener
- AgentParameters
- AgentsChangedListener
- BaseConfig
- ConfigDB
- ConfigEventListener
- Controller
- ControllerDispatcherTarget
- CurveAssignListener
- CurveListener
- DataSourceParameters
- DezentenUI
- Evaluation
- HostgroupEventListener
- ProcessOutputReader
- RegisteredAgentManager
- SimulationEventListener
- SimulationStartupHandler

B.9. Simulation.DbWrapper

- Curve
- Host
- Hostgroup
- SimulationConfig

B.10. Simulation.GUI

- AgentAssignPanel
- AgentCurveAssignPanel
- AgentTableModel
- CbxSelContainer
- ConfigurationCbxModel
- ConfigurationPanel
- CurveDataPanel
- CurveDesignPanel
- CurvePanel
- CurveSelectionPanel
- DBSettingsDialog
- EvaluationPanel
- GuiConfigManagement
- HostgroupCbxModel
- HostgroupPanel

- MainFrame
- OfferTableModel
- OnlineEvalPanel
- SimulationSettingsDialog
- StartupControlPanel
- StatusPanel

B.11. Technics

- ConsumerTechnics
- ProducerTechnics
- Simu
- Technics

C. Datenbankmodell

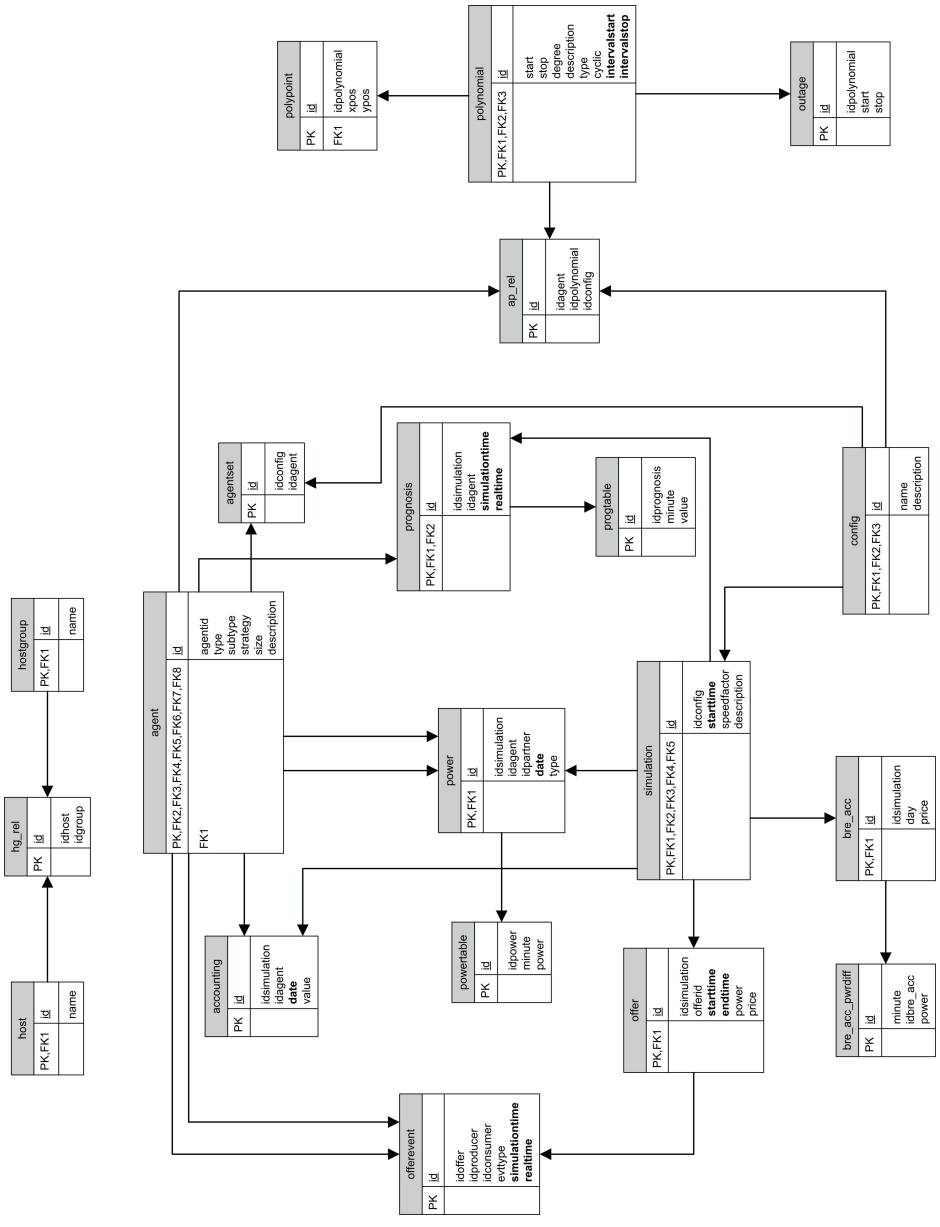

Abbildung C.1.: Modell der Datenbank

Verwendete Software

[1] Cup. http://www.cs.princeton.edu/~appel/modern/java/CUP/.

[2] Eclipse. http://www.eclipse.org.

[3] Gnuplot central. http://www.gnuplot.info/.

[4] Jlex. http://www.cs.princeton.edu/~appel/modern/java/JLex/.

[5] Junit. http://www.junit.org/.

[6] `log4j`. http://jakarta.apache.org/log4j/docs/index.html.

[7] Mysql. http://www.mysql.com/.

[8] Eclipse – omondo uml plugin. http://www.eclipseuml.com/.

Literaturverzeichnis

[Att00] Ammar Attoui: *Real-Time and Multi-Agent Systems*. Practitioner Series. Springer, London; Berlin, 2000.

[Bar94] M. Barabanov: *A Linux-Based Real-Time Operating System*. Diplomarbeit, Dept. of Computer Science, Institute of Technology, New Mexico, 1994.

[Bit99] Rainer Bitsch: *Neue Energieversorgungskonzepte – Intelligente Verteilungsysteme*. In Hans-Jürgen Haubrich (Hg.), *Verteilungsnetze mit dezentralen Stromerzeugungsanlagen*, Bd. 78 von *ETG-Fachbericht*, Seiten 131–136. VDE Verlag, Berlin, 1999.

[Böh00] Sabine Böhm: *Distributed Transaction Management in Safety-Critical Systems*. Diplomarbeit, University of Dortmund, Fachbereich Informatik, 2000.

[Dan92] D. C. Daniels: *The Design and Analysis of Protocols for Distributed Resource Scheduling under Real-Time Constraints*. Dissertation, Wayne State University, June 1992.

[env03] enviva (Hrsg.): *Liberalisierung*, Zugriffsdatum: 21. August 2003.
http://www.envia.de/enviawelt/energieknowhow/357.html

[Exc03] European Energy Exchange (Hrsg.): *EEX Marktmodell*, Zugriffsdatum: 21. August 2003.
http://www.eex.de/futures_market/info/market_model/index.asp

[Fer01] Jacques Ferber: *Multiagentensysteme*. Addison-Wesley, 1. Aufl., April 2001.

[FHI+03a] Frederic A. Folwaczny, Markus Heinz, Dragan Isakovic, Martin Krogmann, Stefan Nobis, Ralf Paaschen, Martin Piayda, Michael Rudermann, Andreas Schröder, Carsten Sommer, Gerd Terhardt, Andreas Volgmann, Horst F. Wedde (Betreuer), Frank Thorsten Breuer (Betreuer) und Wolfgang Freund (Betreuer): *Seminarband der Projektgruppe 432* DEZENTEN . Lehrstuhl III, Universität Dortmund, Apr. 2003.

[FHI+03b] Frederic A. Folwaczny, Markus Heinz, Dragan Isakovic, Martin Krogmann, Stefan Nobis, Ralf Paaschen, Martin Piayda, Michael Rudermann, Andreas Schröder, Carsten Sommer, Gerd Terhardt, Andreas Volgmann, Horst F. Wedde (Betreuer), Frank Thorsten Breuer (Betreuer) und Wolfgang Freund (Betreuer): *Zwischenbericht der Projektgruppe 432* DEZENTEN . Lehrstuhl III, Universität Dortmund, Sept. 2003.

[Fri99] Dr.-Ing. Stefan Fricke: *Werkzeuggestützte Entwicklung kooperativer Agenten im Dienstkontext*. Dissertation, Technische Universitaet Berlin, Fachbereich 13 - Informatik, 1999.

[FSM] FSMLabs: *FSMLabs – The RTLinux Company*. http://www.fsmlabs.com

[Gau97] Arnim J. Gaul: *Wirtschaftlich optimale Laststeuerung mit evolutionären Strategien*. Dissertation, Universität Dortmund, Fakultät Elektrotechnik, 1997.

[Har02] Thomas Hartkopf: *Brennstoffzellen bilden „virtuelle Kraftwerke“. VDE dialog*, Seite 21, Juli/August 2002.

[HBP02] P. J. 't Hoen, D. D. B. van Bragt und J. A. la Poutré: *Bidding with Decommitment in a Multi-Agent Transportation Model*.

Techn. Ber., Centrum voor Wiskunde en Informatica, Amsterdam, NL, November 2002.

[HD94] Klaus Heuck und Klaus-Dieter Dettmann: *Elektrische Energieversorgung*. Vieweg Verlag, Wiesbaden, 3. Aufl., 1994.

[HD01] Klaus Heuck und Klaus-Dieter Dettmann: *Elektrische Energieversorgung*. Vieweg, Braunschweig, 4. Aufl., 2001.

[Hou03] Michael Houben (Hrsg.): *Brennstoffzellen verändern den Markt!*, Zugriffsdatum: 21. August 2003. http://www.mhouben.de/Div/brennstoffzellen.html

[Hug03] Rolf Hug (Hrsg.): *Solarserver*, Zugriffsdatum: 21. August 2003. http://www.solarserver.de/wissen/photovoltaik.html

[Kim02] H.K. Kim: *A TMO Based Approach to Structuring Real-Time Agents*. In *Proc. ICTAI 2002 (IEEE CS 14th Int'l Conf. on Tools with AI)*, Seiten 165–172. Washington, D.C., November 2002.

[Kru99] Philippe Kruchten: *The Rational Unified Process*. Addison-Wesley, 1999.

[LB78] H. Lycklama und D.L. Bayer: *The MERT operating system*. In *Bee System Technical Journal, 57(6)*, Seiten 2049–2086. 1978.

[Lin99] Jon A. Lind: *Realisation of the Highly Integrated Distributed Real-Time Safety-Critical System Melody*. Dissertation, University of Dortmund, Fachbereich Informatik, Fachbereich Informatik, 1999.

[LZ01] Dr. Christoph Lattemann und Pascal Zuber: *Eine marktmikrostrukturtheoretische Analyse der deutschen Strombörsenlandschaft. ZfE – Zeitschrift für Energiewirtschaft*, Bd. 25, Nr. 2, Seiten 75–87, 2001.

[M+] P. Mantegazza et al.: *RTAI API-Dokumentation*. http://www.aero.polimi.it/RTAI/

[Mar00] Peter Marwedel: *Scriptum Prozessrechnertechnik/Eingebettete Systeme*, 2000. Begleitmaterial zur Vorlesung am LS XII.

[MT98] Jochen Markard und Bernhard Truffler: *Ökostrom im Wettbewerb - Nachhaltige Energieerzeugung als Marktfaktor am Beispiel der Wasserkraft*, Okt. 1998.

[MT02] Jochen Markard und Bernhard Truffler: *Dezentrales Energie-Management-System DEMS*, 2002.

[QG00] Volker Quaschning und Michael Geyer: *Einsatzmöglichkeiten regenerativer Energien für eine klimaverträgliche Elektrizitätsversorgung in Deutschland*, 2000. http://www.volker-quaschning.de/downloads/sonnenforum2000_1.pdf

[Rie97] Eike Riedemann: *Softwaretestmethoden*. Teubner, 1997.

[Rip] Ismael Ripoll: *Linuxdevices.com*. http://www.linuxdevices.com/files/misc/ripoll-rtl-v-rtai.html

[Scha] Heiko Schepperle: *Seminar – Multi-Agenten-Systeme (MAS) im Einsatz – Grundlagen und Begriffsbildung*. http://www.schepperle.de/de/papers/Seminar-Multiagentensysteme/Multiagentensysteme_Grundlagen_Ausarbeitung.pdf

[Schb] Peter Schmutter: *Eine Einführung in Multiagentensysteme und deren Verwirklichung im Roboterfußball*. http://www.multiagentsystem.com

[Sta98] John A. Stankovic: *Misconceptions About Real-Time Computing. Computer*, October 1998.

[T+] Linus Torvalds et al.: *The Linux Kernel Archives*. http://www.kernel.org

[Tea] RTAI Development Team: *DIAPM RTAI – Realtime application Interface*. http://www.aero.polimi.it/~rtai/

[Vaz] Xose Vazquez: *Ports of Linux*. http://perso.wanadoo.es/xose/linux/linux_ports.html

[VDE97] *VDE 0105 Teil 1, Betrieb von elektrischen Anlagen*, Oktober 1997. Deutsche Fassung EN 50110-1.

[VDI02] *VDI 6012, Dezentrale Energiesysteme im Gebäude*, Juli 2002.

[WAH+90] H. F. Wedde, G. S. Aljiani, D. Huizinga, G. Kang und B.-K. Kim: *MELODY: A Completely Decentralized Adaptive File System for Handling Real-Time Tasks in Unpredictable Environments*. In *Real-Time Systems Vol. 2 No. 4*. 1990.

[WB00] Horst F. Wedde und Sabine Böhm: *Adaptive distributed real-time transaction management in safety-critical systems*. In *25th IFAC Workshop on Real Time Programming – WRTP'2000*, Seiten 85–91. Euromicro, IFAC, Palma de Mallorca, Spain, 2000.

[WBF01a] Horst F. Wedde, Sabine Böhm und Wolfgang Freund: *Adaptive Concurrency Control in Safety-Critical Real-Time Systems*. In *IFAC Conference on New Technologies in Computer Control 2001 (NTCC'01)*. IFAC, Hong Kong, ROC, November 2001.

[WBF01b] Horst F. Wedde, Sabine Böhm und Wolfgang Freund: *Real-Time Transactions Need Their Constituting Tasks*. In *IFAC Conference on New Technologies in Computer Control 2001 (NTCC'01)*. IFAC, Hong Kong, ROC, November 2001.

[WD91] H. F. Wedde und D. C. Daniels: *Distributed Ressource Scheduling under Real-Time Constraints*. In *Second Great Lakes Computer Science Conference*. Kalamazoo, Michigan, USA, 1991.

[WF00] Horst F. Wedde und Wolfgang Freund: *Harmonious internal clock synchronization*. In *EUROMICRO Workshop on Real-Time-Systems 2000 (ERTS'00)*, Seiten 175–182. Euromicro, IEEE Computer Society Press, Stockholm, Sweden, June 2000.

[WK99] H.-J. Wagner und H. Klein: *Potentiale und Trends dezentraler Elektrizitätserzeugung in Deutschland*. In Hans-Jürgen Haubrich (Hg.), *Verteilungsnetze mit dezentralen Stromerzeugungsanlagen*, Bd. 78 von *ETG-Fachbericht*, Seiten 7–15. Universität GH Essen, VDE Verlag, Berlin, 1999.

[WL97] Horst F. Wedde und Jon A. Lind: *Building Large, Complex, Distributed Safety-Critical Systems. Real-Time Systems*, Bd. 13, Nr. 3, November 1997.

[WL98] Yu-Chung Wang und Kwei-Jay Lin: *Providing Real-Time Support in the Linux Kernel*. Techn. Ber., Department of Electrical and Computer Engineering, University of California, Irvine, California, 1998.

[WLE94] H. F. Wedde, J. A. Lind und A. Eiss: *Achieving Dependability in Mission-Critical Operation Systems Through Adaptability and Large-Scale Functional Integration*. In *Proc. of the ICPADS '94 International Conference on Parallel and Distributed Systems*. Taipei, Taiwan, December 1994.

[WLE95] H. F. Wedde, J. A. Lind und A. Eiss: *Incremental Experimentation: A Methodology for Designing and Analyzing Distributed Safety-Critical Systems*. In *EUROMICRO '95 Workshop on Real-Time Systems*. Odense, Denmark, 1995.

[WSL96] H. F. Wedde, C. Stange und J. A. Lind: *Integration of adaptive file assignment into distributed safety-critical systems*. In *In 21st IFAC/IFIP Workshop on Real-Time Programming*. IFAC, Gramado, RS, Brazil, November 1996.

[Yag01] Karim Yaghmour: *RTAI The Real-Time-Application-Interface*, 2001. http://www.opersys.com/publications/rtai-ols-2001.pdf

[Yod99] Victor Yodaiken: *The RTLinux Manifesto*. Techn. Ber., Department of Computer Science, New Mexico Institute of Technology, New Mexico, March 1999.