BOSCH Technik fürs Leben

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

E.ON Energy Research Center

RWTH AACHEN UNIVERSITY

Abschlussbericht

AGENT

Agentensysteme zur intelligenten und robusten Steuerung komplexer Energiesysteme in Nichtwohngebäuden als Bestandteil des übergeordneten Energiesystems

Förderkennzeichen 03ET1495
Berichtszeitraum: 01.05.2019 - 31.12.2022
Aachen, Juni 2023

RWTH Aachen University, E.ON Energy Research Center,
Lehrstuhl für Gebäude- und Raumklimatechnik
Mathieustrasse 10, 52074 Aachen

RWTH Aachen University, E.ON Energy Research Center,
Institute for Automation of Complex Power Systems
Mathieustrasse 10, 52074 Aachen

Friedrich-Alexander-Universität Erlangen-Nürnberg,
Lehrstuhl für Regelungstechnik
Cauerstr. 7, 91058 Erlangen

Robert Bosch GmbH, Corporate Sector Research and Advance Engineering,
Advanced Energy Systems (CR/AES)
Robert-Bosch-Campus 1, 71272 Renningen

Gefördert durch:

Bundesministerium für Wirtschaft und Klimaschutz

aufgrund eines Beschlusses des Deutschen Bundestages

Autoren

Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren

RWTH Aachen University, Lehrstuhl für Gebäude- und Raumklimatechnik:
Steffen Eser, Thomas Storek, Fabian Wüllhorst, Alexander Kümpel, Rita Streblow, Dirk Müller

RWTH Aachen University, Institute for Automation of Complex Power Systems:
Katharina Wehrmeister, Antonello Monti

Friedrich Alexander Universität, Lehrstuhl für Regelungstechnik:
Thore Wietzke, Knut Graichen

Robert Bosch GmbH, Zentralbereich Forschung und Vorausentwicklung:
Jan Gall, Martin Löhning

Danksagung

Das Forschungsprojekt AGENT wurde in der Zeit vom 01.05.2019 bis zum 31.12.2022 durch den Projektträger Jülich (PTJ) unter dem Förderkennzeichen 03ET1495 betreut. Die Finanzierung erfolgte aufgrund eines Beschlusses des Deutschen Bundestages durch das Bundesministerium für Wirtschaft und Klimaschutz, wofür wir uns herzlich bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Kurzdarstellung des Projekts

1 Aufgabenstellung und Kurzfassung

Durch die fortlaufende Integration neuer Technologien in Gebäude soll der Nutzerkomfort verbessert, sowie der CO_2-Ausstoß von Gebäuden reduziert werden. Im Nichtwohnbereich beinhalten diese Technologien beispielsweise Bauteilaktivierungen, Geothermiefelder, Wärme- und Kältebereitstellung, sowie digitale Steuerungsmöglichkeiten für die Nutzer. Einige dieser Technologien sind schwieriger zu regeln als konventionelle Technologien. Beispielsweise sind Bauteilaktivierungen träger als Radiatoren, und die Arbeitszahl von Wärmepumpen variiert stärker, als der Wirkungsgrad eines Kessels. Kälte- und Wärmebereitstellung können gegeneinander arbeiten, wenn sie schlecht geregelt sind, aber ergänzen sich, wenn sie sowohl als Quelle als auch als Senke eines Kältekreislaufs genutzt werden. Eine intelligente Systemregelung, die die Wechselwirkung einzelner Komponenten untereinander berücksichtigt, ist häufig nicht vorhanden, und erscheint als schwer realisierbar mit herkömmlichen regelbasierten Verfahren. Im Rahmen des Projekts AGENT wurde eine Methodik entwickelt, die einen Gebäudebetrieb auf Basis eines Agentensystems realisiert. Ein Agent ist dabei eine Softwareeinheit, die proaktiv eigene Ziele verfolgt, auf seine Umgebung reagiert, und mit anderen Agenten kommuniziert, um einen effizienten Betrieb des Gesamtsystems zu gewährleisten. Durch die Kommunikation der Agenten kann die Wechselwirkung verschiedener Subsysteme und Komponenten berücksichtigt werden, und durch standardisierte Schnittstellen, soll die Systemgröße durch einfaches Hinzufügen von Agenten skalierbar sein. Die Entwicklung eines einzelnen Agenten soll nur den Agenten selbst sowie seine Schnittstellen enthalten, wodurch Wissen über das Gesamtsystem nicht benötigt wird, und die Komplexität beherrschbar bleibt. Um dieses Ziel zu erreichen, wurde im Rahmen des Projekts eine Python-Bibliothek erstellt, in der standardisiert Agenten entwickelt werden können, und über verschiedene Protokolle miteinander kommunizieren. Verschiedene verteilte Optimierungsverfahren wurden verglichen, um einen optimalen agentenbasierten Gebäudebetrieb zu ermöglichen. Schließlich wurde das Verfahren an einem Demonstrationsgebäude und einem Prüfstand angewandt. Besonders am Demonstrationsgebäude wurde ersichtlich, dass der Aufwand, neue Räume und Komponenten zum Agentensystem hinzuzufügen überraschend gering war, nachdem das initiale Setup vollzogen wurde. Ebenfalls konnte die Wirksamkeit der verteilten Optimierung in der Praxis gezeigt werden. Es wurden jedoch auch einige Schwächen ersichtlich, insbesondere mit Hinblick auf eine notwendige Modellierung von einzelnen Subsystemen. Ebenso muss weiter untersucht werden, inwiefern die verteilte Optimierung robust genug ist, und wo darauf verzichtet werden kann.

2 Voraussetzungen

Das Projekt „AGENT“ wurde als Verbundprojekt von der RWTH Aachen, der FAU Erlangen-Nürnberg und Bosch bearbeitet. An der RWTH Aachen waren der Lehrstuhl für Gebäude- und Raumklimatechnik – EBC, und das Institut Automation of Complex Power Systems – ACS beteiligt. An der FAU Erlangen-Nürnberg war der Lehrstuhl für Regelungstechnik beteiligt. Das EBC hat Kompetenzen im Bereich Modellierung und Regelung von Gebäuden, sowie Monitoring und Infrastruktur für Gebäudeautomation. Das ACS brachte Erfahrung im Bereich Cloud-Computing und Infrastruktur ein. Die FAU unterstützte das Vorhaben im Bereich numerischer Optimierungsverfahren und Regelungstechnik. Letztlich lieferte Bosch wertvolle Erfahrung im Bereich Softwareentwicklung, Regelungstechnik und bot außerdem ein Versuchsobjekt auf dem hauseigenen Forschungscampus.

Zum Projektbeginn gab es viele verschiedene Ansätze zur Umsetzung einer agentenbasierten Programmstruktur, jedoch keine offensichtliche Lösung mit Hinblick auf eine Gebäuderegelung. Viele der untersuchten Plattformen waren veraltet und schlecht gewartet. Vorhandene Energiemanagementsysteme boten keine fortschrittlichen Regelungen an, und basieren häufig auf älterer Technik. Vor diesem Hintergrund wurden die Prototypen des Agentensystems im Projekt neu in Python entwickelt, was den Einbezug vieler moderner externer Werkzeuge ermöglicht.

3 Planung und Ablauf des Vorhabens

Das Verbundvorhaben wurde in die folgenden zehn Arbeitspakete untergliedert, deren zeitlicher Ablauf in Abbildung 3.1 zu sehen ist. Der Ablauf der einzelnen Arbeitspakete lief wie unten beschrieben.

AP1 - Aufarbeitung der Vorarbeiten, grundsätzliche Systemarchitektur sowie technische Grundlagen und Anforderungen

Sämtliche geleisteten Vorarbeiten der Projektpartner wurden durch die Projektbearbeiter zusammengetragen, um einen optimalen Know-How-Transfer zwischen den Projektpartnern sicherzustellen. Aus den Vorarbeiten wurden anschließend grundsätzlich mögliche Systemarchitekturen für ein in der Praxis einzusetzendes Multi-Agenten-System (MAS) abgeleitet. Hierbei wurden Anforderungen an die Softwarestruktur, an Kommunikationsinfrastruktur und Regelungsalgorithmen identifiziert.

AP2 - Entwicklung der Modellbibliotheken

Im AP2 sollten einfach parametrierbare Subsystemmodelle in Modelica entwickelt werden. Im Laufe des Projekts stellte sich heraus, dass Modelica nicht gut geeignet ist, um Modelle für die Optimierung zu entwickeln. Im Rahmen des Vorhabens wurden für die optimierungsbasierten Regelungsverfahren Modelle in CasADi entwickelt. Ebenso wurden Simulationsmodelle in Modelica entwickelt, an denen die Regelungsverfahren getestet wurden, bevor sie im Experiment angewandt wurden.

Dabei wurden bestehende, quelloffene Bibliotheken um Modelle erweitert. Um agentenbasierte Ansätze bereits in dem Software-in-the-Loop Framework realitätsnah evaluieren zu können, wurde weiterhin eine neue quelloffene Bibliothek zur modularen Simulation von Gebäudeenergiesystemen entwickelt.

AP3 - Entwicklung eines Software-in-the-Loop-Frameworks

In AP3 wurde ein Python-Basiertes Framework unter dem Namen AgentLib entwickelt, mit Hilfe dessen sowohl Simulationen, als auch Experimente durchgeführt werden konnten.

Durch die Modularität der entwickelten AgentLib kann das Software-in-the-Loop-Framework auf ein stufenbasierten Ansatz zurückgreifen, welcher eine granulare Steigerung des Komplexitätsgrads Richtung realer Anwendung ermöglicht.

AP4 - Entwicklung der Agenten

In diesem Arbeitspaket wurden die verschiedenen Software-Einheiten (Agenten) zur Regelung des Gesamtsystems entwickelt und implementiert. Dazu gehört vor allem die Weiterentwicklung der AgentLib, die die flexible Entwicklung und Provisionierung der Agenten ermöglicht.

Jeder Agent ist aus Softwaremodulen zusammengesetzt, die jeweils entweder in der AgentLib enthalten sind oder vom Nutzer als individuelle Erweiterung bereitgestellt werden. Innerhalb eines Agenten kommunizieren die Module über einen zentralen Broker.

AP5 - Begleitende Simulationen und Experimente

Die in AP3 entwickelte Software-in-the-Loop-Entwicklungsumgebung wurde im Rahmen von AP5 getestet und in Betrieb genommen. Verschiedene Regelungsansätze wurden simulativ getestet und die Ergebnisse publiziert. Die Inbetriebnahme und Regelung eines HiL-Prüfstandes in Aachen wurde basierend auf der AgentLib-Bibliothek getestet.

AP6 - Nutzermodelle und adaptive Komfortagenten

In AP6 wurden Module für die AgentLib erstellt, die für Nutzerkomfort verantwortlich sind. Darin wird z.B. die Regelung zur Kühlung eines Raums bestimmt, die nötig ist um ein zulässiges Temperaturband einzuhalten. Dieses Temperaturband wird adaptiv basierend auf der Außentemperatur bestimmt, die von einem separaten Wetteragenten zur Verfügung gestellt werden. Die Berechnung der nötigen Kühlung bezieht den erwarteten Wärmeeintrag in das System mit ein.

Diese Module können entsprechend der modularen Struktur als Teil der Konfiguration eines Agenten eingebunden und verwendet werden.

AP7 - Systemarchitektur und Prototypentwicklung

Basierend auf der Festlegung der Entwicklungsumgebungen in AP2 und AP3 sowie den Ergebnissen aus AP5 wurde in diesem Arbeitspaket der Aufbau des Gesamtagentensystems realisiert. Hierbei wurde besonderes Augenmerk auf einfache Konfiguration und Erweiterbarkeit der Struktur gelegt.

Für die Ausführung des Agentensystems wurde die Plattform cloneMAP (cloud-native Multi-Agent Platform) verwendet. Diese wurde von der RWTH im Rahmen des BMBF Projekts ENSURE entwickelt. Der Fokus bei der Entwicklung lag auf einer hohen Skalierbarkeit sowie Ausfallsicherheit der Plattform.

AP8 - Implementierung/Demonstration in einem realen Gebäude inklusive Monitoring

Das entwickelte Multi-Agenten-System wurde auf dem Gelände von BOSCH in Renningen demonstriert. Dabei wurden alle im Projekt entwickelten Komponenten - von der AgentLib, den entwickelten Regelungsalgorithmen, sowie der Cloud-Architektur zusammengefasst und für die Regelung einer Demonstratoretage genutzt.

AP9 - Systematische Analyse, Bewertungskriterien und Vergleich mit vorhandenen Lösungen

AP9 diente der systematischen Evaluierung der Projektergebnisse. Hier wurden Ergebnisse visualisiert und verglichen sowie der Rechen- und Kommunikationsbedarf der entwickelten Lösungen untersucht.

AP10 - Verbreitung und Verwertung der Ergebnisse sowie Projektmanagement

AP10 hat von Beginn des Projektes an die Verbreitung seiner Ergebnisse begleitet. Dazu gehörten sowohl Open-Access-Veröffentlichungen von Journal- und Konferenzbeiträgen, als auch Open-Source-Veröffentlichungen von Ontologie und Modellen.

Zudem wurde in diesem AP die kontinuierliche Zusammenarbeit zwischen den Projektbeteiligten organisiert und ein hohes Maß an Koordination sichergestellt.

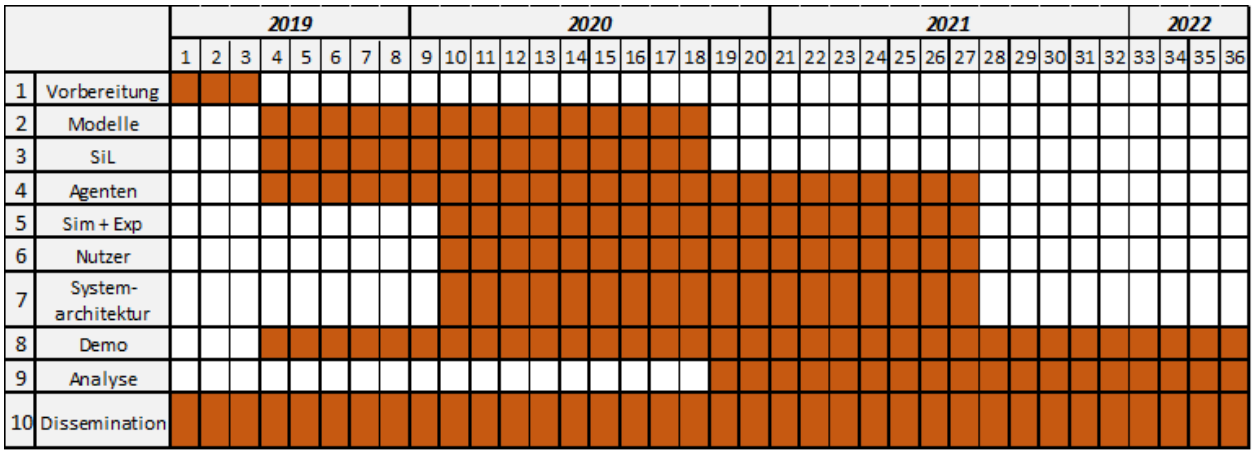


Abbildung 3.1: Balkenplan für das Projekt AGENT

4 Stand der Technik vor Projektbeginn

In der täglichen Praxis des Anlagenbaus im Nichtwohnbereich stehen bereits in der Planungsphase Simulationswerkzeuge zur Verfügung, um die Auswahl der richtigen Systemkomponenten, ihre Dimensionierung und das Zusammenspiel bei dynamisch wechselnden Lasten zu analysieren. So können bereits im Vorfeld des eigentlichen Bauens der Energieinfrastruktur Optimierungen für die geplante Anlagetechnik zur Gebäudeversorgung vorgenommen werden. Besondere Bedeutung gilt dabei der Abbildung und Prüfung der geplanten Regelalgorithmen. Alle Regelungsparameter müssen ermittelt und überprüft werden, um verschiedene Anlagenteile aufeinander abzustimmen.

In vielen Fällen wird dieser Aufwand nicht in einem ausreichenden Maße betrieben und die Betrachtungen beschränken sich auch bei komplexen Systemen auf einfache „statische“ Methoden. Oft fehlt daher bei der Planung das Verständnis und die Erfahrung für die komplexen Interaktionen, die insbesondere in energieoptimierten Systemen durch Vernetzung von Teilsystemen (Abwärmenutzung, freie Kühlung etc.) beachtet werden müssen.

Die heutige Gebäudeautomation besteht hauptsächlich daraus, dass verschiedene Anlagen mit ihrer herstellereigenen Regelung verbaut, und über ein Bussystem an die Gebäudeleittechnik (GLT) angeschlossen werden. Integrierte Zähler- und Datenerfassungssysteme bieten die Grundlage, die Verbräuche zu erfassen und im besten Falle mittels eines Energiemanagementsystems kontinuierlich zu analysieren. Nur durch eine gezielte Analyse dieser Daten kann die Effizienz des Systems langfristig in einem Lernprozess verbessert werden. Der Einsatz einer Gebäudeleittechnik als zentrales Element für alle Mess-, Steuer-, und Regelfunktionen ist heute Stand der Technik in komplexen Anlagen. Die Hauptaufgabe der GLT ist die Visualisierung und Automatisierung der technischen Vorgänge innerhalb des Gesamtsystems. Eine intelligente Systemregelung, die die Wechselwirkung einzelner Komponenten untereinander berücksichtigt, ist jedoch häufig nicht vorhanden oder basiert auf einfachen Regeln, die aus Erfahrungswerten ermittelt wurden.

Beispielhafte Aspekte, die von der GLT optimiert werden sollten sind nachstehend aufgeführt:

- Kontrolle von Ein- und Ausschaltzeitpunkten der Systeme.
- CO_2-basierte Regelung der Raumluftqualität.
- Vermeidung des gleichzeitigen Betriebs von Heizung und Kühlung in Räumen.
- Prioritätenregelung bei mehreren Wärme- oder Kälteerzeugern.
- Energiemonitoring sowie automatisiertes Energiecontrolling.

Neben der Betrachtung von Heizung, Lüftung und Kälte werden heute auch die Lichtsteuerung, die

Beschattung, die Präsenz und die Brandmeldung einbezogen. Bei einigen Systemen können Daten bis hin zur Berücksichtigung einzelner elektrischer Verbraucher betrachtet werden.

Bei herkömmlichen Gebäudeleitsystemen werden alle Datenpunkte zentral gesammelt, und über ein extra für das Gebäude entwickeltes Programm können Entscheidungen getroffen werden. Die Nachteile des zentralen Ansatzes sind unter anderem:

- Hoher Verkabelungsaufwand durch die zentrale Steuerung der Sensoren und Aktoren.
- Anfälligkeit für Störungen mit Auswirkungen auf das gesamte System.
- Hoher Kostenaufwand.
- Änderungen sowie Erweiterungsmöglichkeiten sind sehr eingeschränkt bzw. erfordern hohen Aufwand.
- Leistungsbegrenzung durch Serverschnittstellen.
- Eine Flexibilisierung des Systems (Smart Building) ist nur über einen zentralen Zugriff auf Komponenten (ggf. sogar von extern) möglich, mit den entsprechenden Datenschutzrisiken und einer enormen Komplexität dieser zentralen Optimierung.
- Keine übergeordnete Optimierung über Handelsmechanismen möglich.

In den letzten Jahren wurden vermehrt dezentrale Ansätze untersucht, die einen ersten Schritt in Richtung der Agentensysteme darstellen: Bei einer dezentralen Automationsstruktur ist der Gesamtprozess auf einzelne in sich geschlossene Teilprozesse aufgeteilt. Jeder Prozess arbeitet autark und beinhaltet die notwendigen Grundfunktionen, um diesen Teilprozess zu bearbeiten. Durch die Vernetzung aller Sensoren und Aktoren in ein Inter-/Intranet z.B. Ethernet wird die dezentrale Gebäudeautomation immer weiter verbessert. Vorteile der dezentralen Struktur sind:

- Entlastung der zentralen Steuerung von prozessnahen Aufgaben.
- Zentrale verarbeitet nur übergeordnete Management-Aufgaben.
- Anlagennahe Datenaufbereitung reduziert den Datenverkehr auf ein notwendiges Minimum.
- Hohe Leistungsfähigkeit durch Automationseinheiten der Teilprozesse vor Ort.

Trotz der Verbesserung des Betriebes im Gebäude durch die dezentrale Gebäudeautomation bleiben negative Begleiterscheinungen wie ein zentraler Datenspeicher beziehungsweise eine zentrale Kommunikationsschnittstelle, zu der eine entsprechende Verkabelung notwendig ist. Ebenfalls ist bei Störungen oder veränderten Bedingungen weiterhin eine Bearbeitung durch einen Anwender notwendig, der entsprechende Vorgaben trifft. Eine Verbesserung in der möglichen Erweiterung der Anlagen ist durch die dezentrale Automation gegeben, jedoch ist je nach verwendeter Kommunikationssoftware keine beliebige Erweiterung möglich und erfordert die Bearbeitung durch Fachpersonal.

Eine Möglichkeit, diesen oben beschriebenen Problemen entgegenzuwirken, sind sogenannte Software-Agenten. Diese arbeiten autark und besitzen eine eigens implementierte „Intelligenz“, wodurch der

Aufwand und die Kontrolle durch einen Anwender sowie die Notwendigkeit eines zentralen Speichers mit entsprechender Verkabelung reduziert werden. Ebenfalls kann durch die Agenten eine höhere Stabilität und Leistungsfähigkeit des Systems erreicht werden, da sie in vorher definierten Rahmen eigenständige Entscheidungen treffen können und sich auch im Rahmen einer Optimierung an einem Handelssystem beteiligen können.

Ein Multi-Agenten-System (MAS) bezeichnet die Vernetzung mehrerer Agenten, die über standardisierte Schnittstellen miteinander kommunizieren. Ein MAS bietet Vorteile in der Erweiterbarkeit und Robustheit gegen Ausfälle und Störungen. Neue Komponenten können problemlos in das System integriert werden, indem sie sich selbstständig mit Angabe ihrer Eigenschaften anmelden. Potenzielle Ausfälle einzelner Agenten oder auch Kommunikationsstörungen haben nur minimalen Einfluss auf die Funktion des Gesamtsystems, da alle Agenten aufgrund ihrer Intelligenz auch autark arbeiten können.

Der Datenaustausch mit den anderen Agenten ist wesentlich überschaubarer als bei einem gänzlich zentral organisierten System, da für die Optimierungsaufgabe nicht jeder einzelne Datenpunkt zentral zur Verfügung gestellt und analysiert werden muss. Dies ist gerade in einem Umfeld von diversen zu regelnden Systemen, welche wie beispielsweise in größeren Liegenschaften des Nichtwohnbereichs miteinander verknüpft sind, von hoher Bedeutung. Die Herausforderungen aufgrund der Komplexität des Gesamtsystems werden dank der inhärenten Skalierbarkeit des Multi-Agenten-Ansatzes beherrschbar.

Bei einer Umstellung eines vorhandenen Systems auf ein Agentensystem kann für die Erfassung der Energiedaten von Verbrauchsgeräten bereits installierte Messtechnik zur Zustandsüberwachung genutzt werden. Für die meisten Applikationen, also auch für neu zu errichtende Anlagen, reichen einfach zu erfassende Größen, wie Temperaturen oder Wärmemengen, zumeist aus. Hier können beispielsweise bereits aus einer zentralen Vorlauftemperatur und einer Raumtemperatur alle für die zu regelnde Zone notwendigen Annahmen getroffen werden.

Die Geräte können über bereits vorhandene Schnittstellen für eine zentrale Steuerung, auch im Fall der agentenbasierten geregelt werden. Ist die Kommunikation mit neuen Aktoren notwendig, bietet sich eine funkbasierte Kommunikation an.

Im Folgenden wird ein kurzer Überblick über die aktuelle wissenschaftliche Literatur gegeben.

Zhu et al. [2016] geben einen umfassenden Überblick über den Stand der Technik in Agentensystemen. Sie definieren einen Agenten oder „Schwarmsystem“ als eine Gruppe von sich selbst organisierenden Individuen, die eine gemeinsame Aufgabe erfüllen. Der Informationsaustausch zwischen den Agenten kann laut Zhu et al. [2016] mithilfe eines Graphen beschrieben werden, dessen Knoten die Agenten und dessen Kanten den Informationsaustausch zwischen den Agenten repräsentiert. Diese Darstellung impliziert, dass die Agenten nur mit einer Menge direkter Nachbarn kommunizieren. Dies ist ein Ansatz, der auch von Jalal and Rasmussen [2017] im Rahmen ihrer auf „Limited-Communication“, also beschränkter Kommunikation basierenden verteilten Regelung verwendet wird. Dagegen gibt es auch Veröffentlichungen, die eine uneingeschränkte Kommunikation

zwischen sämtlichen Agenten annehmen.

Analogien, die sich einige Autoren zunutze machen, bestehen zur Spieltheorie (Quijano et al. [2017]) und zum Verhalten von Tierschwärmen wie Ameisenkolonien (Silva et al. [2009]) und Wolfsrudeln (Xi et al. [2016]).

Das Ziel in Agentensystem besteht darin, einen Konsens zu erreichen (Zhu et al. [2016]). Die Kostenfunktionen können dabei, wie im Beispiel eines Microgrids, in der Summe der Erzeugungskosten bestehen (Xi et al. [2016]), wobei der Konsens darin besteht, dass die marginalen Kosten sämtlicher Agenten einander entsprechen. Im Gebäudebereich werden vielfach multikriterielle Kostenfunktionen aus monetären Kosten für die Energiewandlung und Kosten für die Abweichung von Komfortbedingungen bestehen. Allgemein wird in der Regelungstechnik oft eine Kostenfunktion definiert, die einen Kompromiss aus hoher Regelgüte und geringer Stellaktivität erreichen soll und unter anderem von Jalal and Rasmussen [2017] verwendet wird.

Um eine garantierte Konvergenz zu einem globalen Optimum der Kostenfunktion des gesamten Agentensystems zu erreichen, ziehen einige Autoren Dekompositionsmethoden von Optimierungsproblemen heran. Weit verbreitete Ansätze für konvexe Optimierungsprobleme sind Subgradientenmethoden und ADMM (Boyd et al. [2007]; Cai et al. [2016]). Cai et al. [2016] weisen beispielsweise die durch Zerlegung des Gesamtproblems erhaltenen Teilprobleme jeweils einem Agenten zu. Auf diese Weise lassen sich ebenfalls dynamische bzw. physikalische Kopplungen (bspw. durch Wärmeströme) entkoppeln und mit Hilfe von ADMM verteilt und prädiktiv lösen (Bestler and Graichen [2017]). Ein ebenfalls für konvexe Probleme geeigneter Ansatz wird von Jalal and Rasmussen [2017] verwendet und basiert darauf, dass Agenten iterativ Kosten- und Störgrößentrajektorien austauschen. Um die gerade im Gebäudebereich oft nicht geltenden vereinfachenden Annahmen wie Konvexität (Cai et al. [2016]) zu umgehen, können auch metaheuristische Optimierungsalgorithmen zerlegt werden. Ein für Agentensysteme prädestinierter Algorithmus ist laut Silva et al. [2009] die Ant-Colony-Optimization (ACO). In dieser werden künstliche Ameisen als Agenten aufgefasst, die über sogenannte Pheromonmatrizen die Attraktivität von Entscheidungspfaden untereinander kommunizieren. Sie zerlegen Probleme aus dem Supply Chain Management, indem sie unterschiedlichen Teilsystemen wie Produktion oder Logistik jeweils einer künstlichen Ameisenkolonie zuweisen. Weitere Beispiele für zerlegte Metaheuristiken werden im Fall der Particle Swarm Optimization (PSO) von Peng and Zhang [2008] und im Fall des Honeybee-Algorithmus von Mentis and Yilmaz [2014] gezeigt.

Im Gebäudebereich wird das Agentenkonzept unter anderem von Mokhtar et al. [2013] angewendet, die die Komponenten eines realen Gebäudes zunächst über Agenten koordiniert haben, in denen einfache Regeln implementiert waren. Diese Regeln lauten beispielsweise:
“Wenn Zonentemperatur $<$ Sollwert $\rightarrow$ Schalte Wärmepumpe ein; sonst schalte Wärmepumpe aus”.

Dieses einfache System wird durch die Einführung eines Klassifizierungsalgorithmus auf Basis eines neuronalen Netzwerks in den einzelnen Agenten erweitert. Basierend auf den Eingangsgrößen gibt dieser Klassifizierer eine Regelentscheidung aus. Eine weitere Veröffentlichung im Gebäudebereich

ist die bereits angesprochene Anwendung von Dekompositionsmethoden wie ADMM von Cai et al. [2016].

5 Zusammenarbeit mit anderen Stellen

Im folgenden werden die Zusammenarbeiten mit anderen Stellen zu projektbezogenen Themen aufgelistet:

- Es wurden Synergien mit den Projekten:
  - N5GEH: National 5G Energy Hub - Einführung zukunftsträchtiger Kommunikationsstandards in der Energietechnik; Teilvorhaben: Einbindung dezentraler Energieversorgungssysteme (FKZ: 03ET1561B);
  - FISMEP - FIWARE für Smart Energy Plattform, Teilvorhaben: Feldtest Demonstration (FKZ: 0350018A);

  genutzt, in denen ebenfalls die FIWARE Plattform verwendet wurde.
- Eine Zusammenarbeit mit dem Forschungszentrum Jülich für Plattform spezifische Themen wurde etabliert.
- Die im Projekt entwickelte Softwarebibliothek *AgentLib* wurde quelloffen auf GitHub veröffentlicht und wird in den Foschungsvorhaben:
  - DZWi - Digitaler Zwilling von Wärmeerzeugersystemen als Wegbereiter für die Entwicklung emissionsarmer Gebäudeenergietechnik (FKZ: 03EN1022B);
  - LLEC: Living Lab Energy Campus (FKZ: 03ET1551A);

  angewendet. Rückmeldungen und Verbesserungsvorschläge sind bei der Entwicklung der Bibliothek berücksichtigt worden.
- Die Entwicklung der verteilten Optimierungsalgorithmen wurde gleichzeitig in den DFG Projekten (Gr 3870/4-1) und (Gr 3870/6-1) vorrangetrieben. Die Erkenntnisse bezüglich den Algorithmen und Anwendungen wurde zwischen den Projekten geteilt.

Teil II

Eingehende Darstellung der Projektergebnisse

6 Projektergebnisse

In diesem Kapitel werden die Projektergebnisse detailliert erläutert. Die Struktur ist lose an die Arbeitspakete angelehnt, wird jedoch zusammengefasst, wo es thematisch sinnvoll ist. Zunächst wird in Abschnitt 6.1 auf Grundlagen und Literatur eingegangen. In Abschnitt 6.2 wird die Modellierung für das Projekt motivert und erläutert. Ein Kernergebnis des Projekts, eine Python-Bibliothek für die Ausführung von Agentensystemen namens *AgentLib* wird in Abschnitt 6.3 eingeführt. Dann wird in Abschnitt 6.4 ein Software-in-the-Loop Konzept vorgestellt, das die Agenten der *AgentLib* einbindet. In Abschnitt 6.5 werden die grundlegenden Agenten, die während des Projekts entwickelt und genutzt wurden vorgestellt. Dann wird im Abschnitt 6.6 noch einmal detaillierter auf verteilte Regelungsalgorithmen, Simulationen mit selbigen und ihre Anwendung in der *AgentLib* eingegangen. Erste experimentelle Ergebnisse der Agentenmethode an einem Prüfstand werden in Abschnitt 6.7 vorgestellt. Letztlich beinhaltet Abschnitt 6.8 die Anwendung aller im Projekt entwickelten Komponenten an einem realen Demonstratorgebäude.

6.1 Vorarbeiten und Literaturrecherche

In einem ersten Schritt des Projektes wurden existierende Kenntnisse und Vorarbeiten im Bereich der Gebäudeautomation, Modellierung und Optimierung gesammelt und aufgearbeitet. Darüber hinaus wurde eine Literaturrecherche durchgeführt, um eine Übersicht bestehender Lösungsansätze für die Umsetzung von Multi-Agenten System (MAS) zu erlangen. Hierauf aufbauend wurden weitergehende Ansätze in Bezug auf Gebäudeenergiesysteme abgeleitet und erarbeitet. Die Ergebnisse werden im Folgenden dargestellt.

6.1.1 Grundlagen Multi-Agenten Systeme

MAS ist ein Computerparadigma, das seit den 1990er Jahren existiert und für viele verschiedene Probleme anwendbar ist, darunter Systemidentifizierung, Modellierung oder Steuerung komplexer Systeme wie intelligente Stromnetze oder Computernetze [Dähling et al., 2021; Dorri et al., 2018; Joe and Karava, 2017] Die praktische Umsetzung von MAS befindet sich jedoch noch im Anfangsstadium [Kravari and Bassiliades, 2015b]. Das Paradigma verwendet unabhängige, lose gekoppelte Softwareeinheiten, die spezifische Funktionen kapseln und miteinander interagieren, um übergreifende oder agentenbezogene Aufgaben zu lösen [Labeodan et al., 2015]. Kurz gesagt, es handelt sich um ein hochmodulares Softwarekonzept, das die Implementierung komplexer Berechnungsaufgaben beschleunigen kann. Darüber hinaus muss ein Agent gemäß Wooldridge and Jennings folgende Kriterien erfüllen:

- *Soziale Fähigkeit*: Agenten können Wissen von anderen Agenten in der Umgebung teilen und anfordern [Dorri et al., 2018; Labeodan et al., 2015]
- *Autonomie:* Basierend auf ihren Zielen treffen Agenten selbstständig Entscheidungen und Handlungen [Dorri et al., 2018; Labeodan et al., 2015]
- *Reaktivität:* Agenten haben ein Feedbackverhalten und können rational sein [Labeodan et al., 2015]
- *Proaktivität:* Jeder Agent nutzt seine Geschichte und Informationen aus seiner Umgebung, einschließlich benachbarter Agenten und erfasster Messungen, um zukünftige Handlungen zu antizipieren. Dies ermöglicht es einem Agenten, nachhaltigere und reflektiertere Entscheidungen zu treffen. [Bakakeu et al., 2017]

Dorri et al. geben einen umfassenden Überblick über MAS, skizzieren ihre Eigenschaften, die für die Kommunikation zwischen Agenten verwendeten Methoden und die Herausforderungen bei der Implementierung von MAS. Hierbei konzentrieren sich die Autoren insbesondere auf die Festlegung eines gemeinsamen Verständnisses und einer allgemeinen Definition von MAS und auf die Abgrenzung von anderen ähnlichen Systemen und nicht auf spezifische Anwendungen von MAS und zeigen verschiedene potenzielle Anwendungen auf. Bakakeu et al. schlagen vor, cyberphysikalische Systeme (CPSs) für Gebäudeenergiesysteme (GESs) zu entwerfen, wobei MAS als Lösung zur Überwindung der durch die Entwicklung höherer Regelungskonzepte und anderer intelligenter Dienstleistungen eingeführten Herausforderungen eingesetzt wird. Im Bereich der Gebäudesteuerung bietet MAS hierbei eine breite Palette von Möglichkeiten für die Umsetzung intelligenter Energiedienstleistungen [González-Briones et al., 2018]. González-Briones et al. geben einen Überblick über bestehende Studien im Bereich der Gebäudeautomation. Die Autoren heben auch Methoden zum Entwurf von MAS wie GAIA hervor. Unter Verwendung von GAIA stellen die Autoren auch einen Prototyp eines MAS-Designs für den potenziellen Einsatz in Gebäuden vor. Zusammenfassend ermöglichen MASs nicht nur einen hochgranularen bausteinartigen Ansatz zur Aufteilung und Beherrschung großer Optimierungsprobleme, sondern bietet auch Effizienz, geringe Kosten, Skalierbarkeit, Ausfallsicherheit, Kapselung herstellerspezifischer Daten und Ad-hoc-Integration in übergreifende Softwarestrukturen für verteilte Energiesysteme [Dorri et al., 2018; Labeodan et al., 2015]. Schließlich machen die Ähnlichkeiten der Konzepte des Internet of Things (IoT) und MASs, MAS zu einem Paradigma, das für die Implementierung von realen Anwendungen und die Umsetzung von höheren Regelungskonzepten für GESs interessant ist [Ahmad et al., 2016; Dorri et al., 2018; Labeodan et al., 2015].

6.1.2 Frameworks für die Erstellung von Multi-Agenten Systemen

Es existieren bereits mehrere öffentliche Software-Frameworks für die Implementierung von MASs. Neben zahlreichen individuellen proprietären Implementierungen sind Java Agent DEvelopment framework (JADE) oder Smart Python Agent Development Environment (SPADE), die in den Programmiersprachen Java bzw. Python geschrieben sind, die wohl bekanntesten Frameworks

[González-Briones et al., 2018; Palanca et al., 2020]. Trotz weiterer Alternativen und der kürzlichen Überarbeitung von SPADE, verwenden die meisten Studien jedoch entweder JADE oder proprietäre Implementierungen [González-Briones et al., 2018; Palanca et al., 2020]. Hierbei wird Java als primäre Programmiersprache verwendet [Kravari and Bassiliades, 2015b]. Ein möglicher Grund für die Verwendung proprietärer Lösungen könnte sein, dass sowohl SPADE als auch JADE für eine allgemeine Implementierung von MAS gedacht sind und keine anwendungsfallspezifischen Komponenten bieten. Weiterhin stützen sie sich stark auf das Extensible Messaging and Presence Protocol (XMPP). Obwohl XMPP Funktionen bietet, die im Allgemeinen die Umsetzung von MAS begünstigen, konzentriert sich ein zu großer Teil der Architektur auf ein komplexes Kommunikationsprotokoll, was ein einfaches Prototyping und Debugging in einem frühen Entwicklungsstadium von MAS erschwert. Hierdurch wird die Nutzung und Übernahme für die Implementierung und Anwendung im Bereich der Energietechnik gehemmt. Neben den programmiersprachenspezifischen Frameworks gibt es auch einige sprachenunabhängige Ansätze wie z. B. cloneMAP (cloudnative Multi-Agent Platform)[1], die das Potenzial von Cloud-Technologien, wie containerbasierte Implementierungen und etablierte Webstandards nutzen [Dähling et al., 2021]. Hierdurch wird erreicht, das MAS hoch skalierbar und fehlertolerant werden [Dähling et al., 2021]. Darüberhinaus ermöglicht die Einbettung von Agenten in Softwarecontainer eine programmiersprachenunabhängige Implementierung der einzelnen Agenten.

6.1.3 Internet of Things Plattformen

Der zeitweise hohe Bedarf an Rechenleistung, die zunehmende Dezentralisierung, die Notwendigkeit der einfachen digitalen Vernetzung diverser digitaler Schnittstellen unterschiedlicher Hersteller erfordern zunehmend den Einsatz von cloud computing und IoT als zugrunde liegende Informations- und Kommunikationstechnologie (IKT) für zukünftige Energiesysteme [Schmidt and Åhlund, 2018; Dähling et al., 2021; Ahmad et al., 2016].

Vor dem Hintergrund der in Unterabschnitt 6.1.1 erwähnten Ähnlichkeiten der Konzepte von IoT und MASs wurde im Projekt auf Ergebnisse untenstehender Forschungsprojekte aus dem Bereich IKT zurückgegriffen:

- Im vom Bundesministerium für Wirtschaft und Klimaschutz (BMWK) geförderten Projekt N5GEH (FKZ: 03ET1561B) sowie im ERAnet Projekt FISMEP (FKZ: 0350018A) wurde bereits an quelloffenen Plattformen für Energiesysteme gearbeitet. Daraus ist eine cloudbasierte Systemarchitektur für die Automatisierung thermischer und elektrischer Energiesysteme hervorgegangen. Weiterhin wurden hier bereits einheitliche Datenstrukturen geschaffen sowie grundlegende Feedback-Regler implementiert.
- Im vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Projekt ENSURE (FKZ: 03SFK1CO) wurde die cloudbasierte Multi-Agenten Plattform cloneMAP entwickelt. Diese basiert auf Techniken des Cloud Computings, um hohe Skalierbarkeit und

[1] Dähling et al.

Fehlertoleranz zu ermöglichen. Die Verwendung von cloneMAP im Projekt AGENT wird in Unterabschnitt 6.3.2 beschrieben.

6.1.4 Virtuelles Testen und Bewerten von Gebäudeautomationssystemen

Ein wesentlicher Schritt bei der Implementierung ist ein umfassender Test der Automatisierungsfunktionen, um die Risiken von Fehlfunktionen und Leistungseinbußen zu reduzieren [Blum et al., 2019; Wetter, 2011]. Einerseits werden in der Praxis oft nur stichprobenartige Tests der Automatisierungsfunktion bei der Inbetriebnahme von der Inbetriebnahme von BAS durchgeführt. Dieses Vorgehen erlaubt in der Regel keine Quantifizierung der Energieeffizienz und die Vorteile von intelligenten Regelungskonzepten werden nicht von vornherein sichtbar. Daher wird die Energieeffizienz oft während des Betriebs bewertet, und die Automationsprogramme müssen gegebenenfalls vor Ort angepasst werden. Dieser Prozess ist nicht nur komplex und langwierig, sondern auch kostenintensiv. Es existieren bereits einige Forschungsarbeiten im Bereich des virtuellen Testens und Bewertens von Gebäudeautomationssystemen. Diese werden nachfolgend vorgestellt.

Blum et al. [2019] und Huang et al. [2018] geben einen umfassenden Überblick über existierende Arbeiten im Bereich des Testens intelligenter Regelungskonzepte für BAS. In der Gebäudesimulation ist das am weitesten entwickelte Rahmenwerk das Building Controls Virtual Test Bed (BCVTB), das von Wetter [2011] vorgestellt wurde. BCVTB hilft nicht nur bei der Entwicklung von Automatisierungsalgorithmen, sondern ermöglicht auch die Echtzeitsimulation sowie die Co-Simulation und den Datenaustausch zwischen Gebäudesimulationsmodellen, die mit unterschiedlichen Simulationsumgebungen entwickelt wurden. Außerdem ermöglicht es die Kopplung von Simulationsprogrammen mit tatsächlicher Hardware. Der letztgenannte Aspekt ist besonders wichtig, da die Leistung intelligenter Regelungskonzepte auch von der zugrunde liegenden Netzwerkinfrastruktur abhängt. Vom Design her kann BCVTB als modulare Middleware zur Kopplung einer beliebigen Anzahl von Simulationsprogrammen und Steuerungsschnittstellen betrachtet werden (Wetter [2011]). Das Framework wurde jedoch seit 2016 nicht mehr weiterentwickelt und umfasst kein Konzept für die Integration mit IoT-Systemen, die in Zukunft für die Gebäudeautomation immer wichtiger werden. Darüber hinaus existiert der vom National Renewable Energy Laboratory (NREL) entwickelte Alfalfastack[2], der bereits ein cloudbasiertes Emulationsframework für Gebäude bereitstellt und EnergyPlus als Simulationsengine verwendet. Letzteres schränkt seine Verwendung für die Entwicklung von dynamischen Steuerungsanwendungen allerdings ein. Blum et al. [2019] stellen den Prototyp des Building Operation Performance Test (BOPTEST)-Framework für das simulationsbasierte Benchmarking intelligenter Regelungskonzepte vor. Sie definieren nützliche Anforderungen für eine virtuelle Testumgebung und beschreiben deren Wert für die Beteiligten. Außerdem stellen sie eine Reihe von üblichen Bewertungskriterien (*engl.* key performance indicators (KPIs)) für die Bewertung und den Vergleich intelligenter Regelungskonzepte vor. Ihr Schwerpunkt der Arbeiten liegt hier jedoch auf der Entwicklung und dem Vergleich intelligenter Regelungskonzepte als auf

[2] https://github.com/NREL/alfalfa

der Unterstützung ihrer praktischen Umsetzung.

Sendorek et al. [2018] stellen ein allgemeines softwaredefiniertes Prüfstandskonzept für IoT-Systeme vor. Sie stellen ein mehrstufiges Testverfahren für IoT-Netzwerke vor, das von einem vollständig softwaredefinierten Szenario bis hin zur Integration realer Hardware reicht, die Signale aus der realen Welt emuliert. Obwohl dieser Ansatz die Bewertung von Netzwerk-Kommunikationsmechanismen ermöglicht, erlaubt er nicht die erforderliche Bewertung von KPIs, wie sie von z. B. Blum et al. [2019] eingeführt wurden. Ähnlich stellen Huang et al. [2018] bereits ein agentenbasiertes Emulations-Framework für Hardware-in-the-Loop zum Testen von Reglern vor, in dem sie ein Co-Simulations-Setup mit Software-Agenten als modulare Schnittstellen zu verschiedenen Simulations-Engines realisieren.

6.1.5 Verteilte und optimierungsbasierte Regelung

Ein Hauptziel des Projektes ist die Entwicklung einer verteilten Steuerung und Regelung von Energiesystemen in Nichtwohngebäuden mittels MASs. In der Literatur wird die modellprädiktive Regelung (MPC) als vielversprechende Regelungsmethodik für GES genannt. MPC scheint die ideale Regelungsstrategie zu sein, um mit den Herausforderungen von GES umzugehen [Afram and Janabi-Sharifi, 2014; Killian and Kozek, 2016]:

- große Speicherkapazitäten;
- technische und thermische Randbedingungen;
- zeitlich veränderliche nichtlineare Systemdynamik wie Schaltaggregate oder Sollwertvariationen;
- zeitlich veränderliche interne und externe Störungen (Belegung, Umgebungstemperatur, Sonneneinstrahlung).

Insbesondere in Gebäuden mit einer großen thermischen Masse ist MPC in der Lage, die Vielzahl unterschiedlicher Zeitkonstanten dieser Systeme zu handhaben, um den Energiebedarf zu optimieren und gleichzeitig den Nutzerkomfort sicherzustellen sowie das große Flexibilitätspotenzial GES als Systemdienstleistung für das elektrische Netz nutzbar zu machen Drgoňa et al. [2020]; Killian and Kozek [2016]. Gegenüber klassischen Regelungsmethoden, wie z. B. On/Off- oder proportional integral derivative (PID)-Reglern, können mittels MPC Energieeinsparung zwischen 20 % und 35 % bei einer gleichzeitigen Reduktion der Betreiberkosten von bis zu 73 % erzielt werden. [Afram and Janabi-Sharifi, 2014; Serale et al., 2018; Killian and Kozek, 2018]

Einen guten Einstieg in die verteilte und optimierungsbasierte Regelung von Systemen bietet Scattolini [2009]. In dem Artikel wird eine Klassifikation von verteilten Ansätzen vorgestellt. Demnach gibt es neben der klassischen zentralen Regelungsstruktur die dezentrale Regelung, bei der es mehrere lokale Regler gibt, welche aber nicht kommunizieren. Weiterhin gibt es die verteilte Regelung, bei der die lokalen Regler miteinander kommunizieren und somit das Verhalten der anderen Regler kennen. Hierdurch kann eine bessere Regelgüte erzielt werden. Eine spezielle Ausprägung einer

verteilten Regelung ist die hierarchische Regelung, bei der die Regler auf verschiedenen Ebenen agieren.

Aufgrund der sehr vielen in der Literatur vorgeschlagenen Ansätze zur verteilten Regelung ist es für die Auswahl eines geeigneten Ansatzes wichtig, die Ansätze mithilfe von Eigenschaften zu strukturieren. In Anlehnung an Scattolini [2009] und Maestre and Negenborn [2014] sind die wichtigsten Eigenschaften

- die Topologie des Kommunikationsnetzwerkes (vollständig verbunden, partiell verbunden),
- der Ursprung der Interaktion (Regler für autonome Systeme kommunizieren für Koordination/Synchronization, ein großes zusammenhängendes System wird in verkoppelte Subsysteme zerlegt),
- Häufigkeit des Austausches von Informationen in einem Abtastschritt (iterativer Algorithmus, nicht-iterativer Algorithmus) und
- Kostenfunktion der lokalen Regler (kooperative Algorithmen bei denen jeder lokale Regler einen Teil der globalen Kostenfunktion miniert, egozentrische Algorithmen welche zu einem Nash-Equilibrium führen).

Weiterhin können die Algorithmen eingeteilt werden durch die Interaktion zwischen den Subsystemen sowie die kommunizierte Information zwischen den lokalen Reglern [Christofides et al., 2013]. Nach Müller and Allgöwer [2017] lassen sich die Algorithmen einteilen in iterative Ansätze, sequentielle Methoden, Ansätze basierend auf Robustheitsbetrachtungen und Algorithmen mit Konsistenzbedingungen.

Eine Möglichkeit zur verteilten modellprädiktiven Regelung von kontinuierlichen nichtlinearen System, stellt der von Bestler and Graichen [2019] vorgeschlagene Algorithmus mit Konsistenzbedingungen dar. Ziel der verteilten Regelstruktur ist, das sogenannte *zentrale Optimierungsproblem* (welches bei modellprädiktiver Regelung in jedem Abtastschritt gelöst werden muss) in kleinere Teilprobleme aufzuspalten. Diese werden auch als *lokale Probleme* bezeichnet. Die lokalen Probleme teilen sich jedoch weiterhin dieselben Variablen (z.B. wegen der zusammenhängenden Systemdynamik), sie sind also gekoppelt. Um dies zu ändern, werden *Kopien* der jeweiligen Variablen sowie *Koordinierungsvariablen* eingeführt. Damit diese konsistent zueinander sind, werden zusätzliche Beschränkungen in den lokalen Problemen hinzugefügt. Somit können die Probleme als Maximierungs-Minimierungsproblem formuliert werden. Zur Lösung wird die Alternating Directions Method of Multipliers (ADMM) verwendet. Um zu gewährleisten, dass die Agenten auf diese Art und Weise Konsens erzielen, werden zwischen den einzelnen Berechnungsschritten die jeweiligen Trajektorien zwischen den lokalen Systemen ausgetauscht.

Die Berechnungsschritte der einzelnen Agenten können somit unabhängig voneinander und daher parallel ablaufen. Zur Synchronisierung der Berechnungs- beziehungsweise Kommunikationsschritte kann beispielsweise ein Koordinierungsagent implementiert werden.

6.2 Entwicklung von Modellbibliotheken

Um MPC für die industrielle Anwendung in GES attraktiv zu machen und die Wirksamkeit zu demonstrieren, sind nach Ceccolini und Sangi Ceccolini and Sangi [2022] umfangreiche Test- und Bewertungsverfahren notwendig. Hierbei können insbesondere modellbasierte Testverfahren zum Einsatz kommen.

Um diese Anforderungen zu erfüllen wurden im Rahmen des Projektes existierende Modellbibliotheken, wie die Modelica-Bibliothek *AixLib* weiterentwickelt sowie ein Modell des Demonstrators in SIMULINK aufgebaut. Darüberhinaus wurden Methoden entwickelt, die eine einfache anwendungsspezifische Anpassung der Modelle ermöglichen.

6.2.1 Integration eines CO_2-Modells in das Reduced-Order-Modell der AixLib

Zur Beurteilung des Nutzerkomforts in klimatisierten Gebäuden ist neben der Temperatur die Luftqualität entscheidend. Ein Indikator für die Beurteilung der Innenraumluft ist die CO_2Konzentration. In vielen modernen Gebäuden gilt die CO_2Konzentration daher als Regelgröße für die Belüftung. Um im Laufe des Projektes implementierte Regelungsalgorithmen unter Berücksichtigung einer akzeptablen CO_2Konzentration simulativ testen zu können, waren entsprechende Gebäudemodelle notwendig.

Die Vorarbeiten der RWTH umfassen verschiedene Gebäudemodelle der quelloffenen Modelica-Bibliothek *AixLib*. Die Modelle bilden bisher eine homogene Temperatur jeder Zone ab und werden im Rahmen dieses Projekts um einen CO_2Knoten erweitert. Zunächst werden hierfür die wesentlichen Einflussfaktoren identifiziert und mathematisch beschrieben. Im Anschluss werden Teilmodelle für die Einflussfaktoren modelliert. Diese Teilmodelle werden in das Reduced Order Modell der AixLib integriert. Im Reduced Order Modell wird die Luft im Raum jederzeit als ideal durchmischt und die Luftdichte als konstant angenommen.

Die wichtigsten Einflussfaktoren für die Änderung der CO_2-Konzentration sind die Luftwechselrate und CO_2-Quellen im Raum. Die Luftwechselrate setzt sich aus Fensteröffnung, natürlicher Infiltration und Ventilation des Lüftungsgeräts zusammen. Als CO_2-Quelle wird die CO_2-Emission der anwesenden Personen mathematisch beschrieben und anschließend als Teilmodell implementiert. Weitere CO_2-Quellen und -Senken werden vernachlässigt. Um die Änderung der CO_2-Konzentration im Raum zu berechnen, wird die CO_2-Masse der modellierten Zone bilanziert. Die Massenbilanz wird als eigenes Teilmodell implementiert. Die zuvor berechnete Luftwechselrate und CO_2-Emission sind die Eingangswerte dieses Teilmodells.

Das Modell ist in Releases der *AixLib* sowie dem Hauptbranch veröffentlicht. Die Qualität wird durch kontinuierliche Tests gesichert.

6.2.2 Entwicklung der Modellbibliothek BESMod

Die detaillierten Modelica-Modelle werden als Emulatoren realer Anlagen verwendet. Durch die Abbildung von nichtlinearitäten, Dynamiken sowie diskreten und kontinuierlichen Ereignissen in allen Domänen von Gebäudeenergiesystemen (Hydraulik, Lüftung, Elektrik, Regelung, Gebäudephysik, Trinkwarmwasser) können die Emulatoren möglichst realitätsnah realisiert werden. Durch die Verwendung von quelloffenen Tools zur Kalibrierung, z.B. Wüllhorst et al. [2022], können die Emulatoren sukzessive validiert werden. Neben der quelloffenen Modelica-Bibliothek *AixLib* existieren weitere quelloffene Bibliotheken mit Fokus auf Gebäudeenergiesysteme. Diese sollten zusammen mit der *AixLib* verwendet werden. Dadurch wird redundante Modellierungsarbeit vermieden.

Um agentenbasierte Konzepte möglichst realitätsnah bewerten zu können, sollten die Emulatoren möglichst detailliert sein und alle relevanten Domänen berücksichtigen. Zum Zeitpunkt der Projektdurchführung hat keine der bestehenden quelloffenen Bibliotheken diese Anforderungen erfüllt. Der Fokus der Bibliotheken lag vor allem auf Komponentenmodellen, wie z.B. die CO_2-Quellen (Vergleich Abschnitt 6.2.1). Daher wurde im Rahmen des Projektes die quelloffenen Modellbibliothek *BESMod* entwickelt. *BESMod* stellt Module für Modelle von Gebäudeenergiesystemen bereit. Dabei werden keine neuen Komponenten modelliert. Stattdessen werden bestehende Bibliotheken, wie z.B. die *IBPSA*, *AixLib*, *Buildings* oder *BuildingSystems*, eingebunden. Durch die modulare Gestaltung entstehen zwei Vorteile für die Bewertung agentbasierter Ansätze: Erstens können typische, wiederkehrende Teilsysteme von Gebäuden wiederverwendet werden. Somit wird Modellierungsaufwand reduziert. Zweitens können Agenten für jedes dieser typischen Module direkt über FMUs eingebunden werden. Somit dient *BESMod* einerseits zur Emulation realer Systeme, andererseits können die Modelle als Prozessmodell des Agenten dienen. Das Schema der modularen Gebäudeenergiesystemsimulation ist in Abbildung 6.1 dargestellt. Jedes Modul kann durch einen Agenten repräsentiert werden. Dies wird durch Icons visualisiert.

Für weiterführende Beschreibungen der Modellbibliothek sei an dieser Stelle auf den Konferenzbeitrag verwiesen [Wüllhorst et al., 2023]. Die Veröffentlichung wurde mit dem Best-Paper Award ausgezeichnet.

6.3 Softwarebibliothek für die Entwicklung von Multi-Agenten System

6.3.1 AgentLib

Ein elementarer Bestandteil des Projektes ist die Entwicklung einer Software-Bibliothek. Die Bibliothek namens *AgentLib* ermöglicht hierbei die flexible Entwicklung und Provisionierung von Softwareagenten.

Die Implementierung von Agenten mit der *AgentLib* folgt dem in Abbildung 6.2 dargestellten modularen Entwurfsmuster, wobei jeder Agent "*on-the-fly*" über eine von extern bereitgestellte Konfiguration instantiiert werden kann. Basierend auf dieser Konfiguration initialisiert der Agent

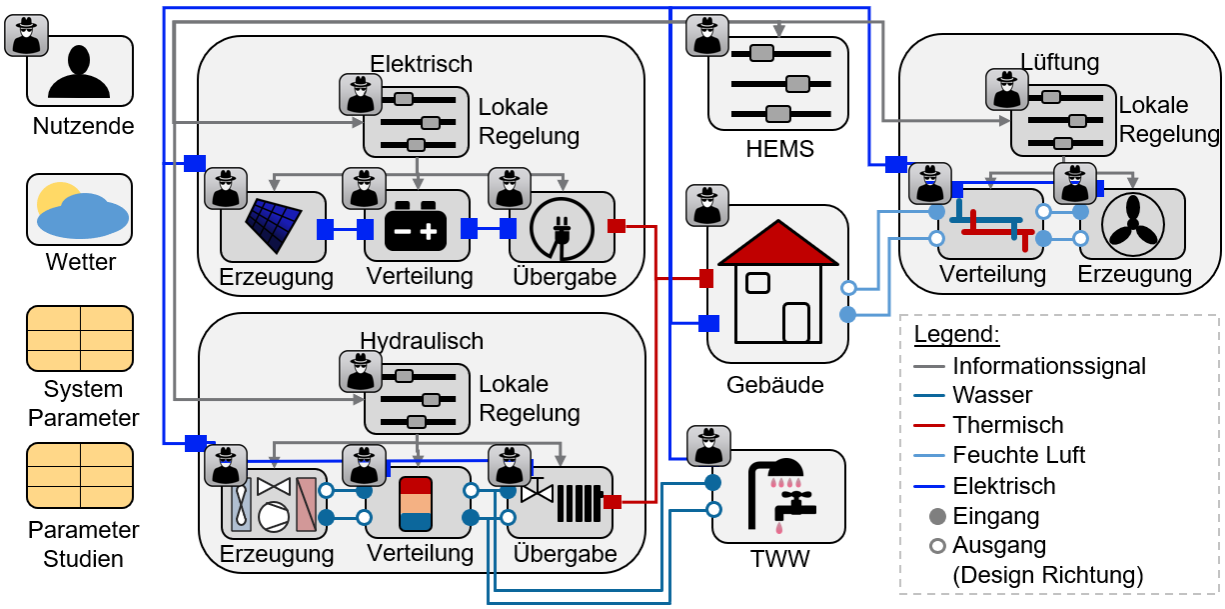


Abbildung 6.1: Struktur der modularen Modell-Bibliothek *BESMod*. Durch die physikalische Modellierung aller relevanten Teilsysteme eines sektorengekoppelten Gebäudeenergiesystems können Regelungsansätze realitätsnah und kostengünstig bewertet werden. Die modulare Struktur ermöglicht die direkte Applikation von Agenten.

Module, die in der Bibliothek enthalten sind oder als individuelle Erweiterungen vom Nutzer bereitgestellt werden. Diese Module können verschiedene Aufgaben übernehmen, beispielsweise die Kommunikation mit anderen Agenten oder mit der Feldebene, Regelungsaufgaben oder individuellere Funktionen. Dazu können Komfortbestimmungen, Datenabfragen aus dem Internet, oder Nutzerinteraktionen zählen.

Ein einzelner Agent kann mehrere solcher Module enthalten. Er benötigt jedoch immer mindestens ein Kommunikationsmodul, um mit seiner Umgebung zu interagieren, und ein weiteres Modul, um eine spezifische Aufgabe zu erfüllen. Innerhalb eines Agenten kommunizieren die Module über einen zentralen *DataBroker*. Für die Kommunikation zwischen und auch innerhalb der Agenten wird eine spezielle Datenstruktur, die sogenannte *AgentVariable*, die auf dem Modelica Association Project "FMI" [2020] basiert, als standardisiertes Nachrichtenaustauschformat verwendet.

DataBroker

Eine wesentliche Komponente innerhalb eines Agentenobjekts bildet der *DataBroker* als zentrale Plattform für den Datenaustausch innerhalb eines Agenten. Alle Variablen, die innerhalb eines Agenten, als auch zwischen mehreren Agenten ausgetauscht werden, passieren den *DataBroker*. Wenn eine Variable durch ein Modul geschrieben wird, werden alle anderen Module, die diese

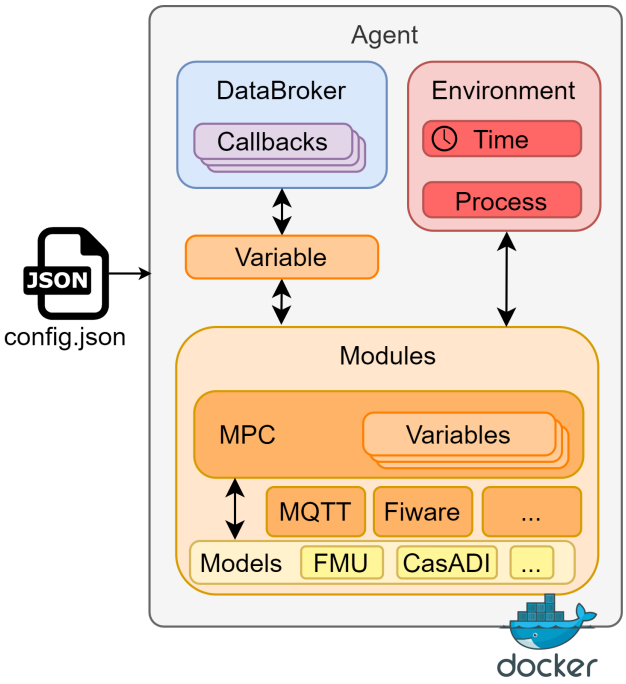


Abbildung 6.2: Struktur eines modularen Software-Agenten der AgentLib

Variable definiert haben, von der Änderung benachrichtigt und erhalten den neuen Wert.

Eine Variable enthält dabei nicht nur einen Wert, sondern hat die in Auflistung 6.1 dargestellten Eigenschaften.

Auflistung 6.1: Definition einer Variablen

```
class BaseVariable(BaseModel):
    name:           str
    alias:          str = None
    type:           str = None
    value:          Any = None
    unit:           str = None
    description:    str = None
    timestamp:      float = None
    source:         Source = None
    shared:         bool = False
    rdf_class:      URIRef = None
```

Dabei ist der *name* entscheidend für die internen Abläufe und die Konfiguration einens Moduls, während der *alias* die öffentliche Bezeichnung einer Variable widerspiegelt und für die Kommunikation zwischen Agenten entscheidend ist. Der *type* einer Variable wird genutzt, um die Klasse des Werts der Variablen wieder zu konstruieren, nachdem der Wert für die Kommunikation als `.json` serialisiert wurde. Die *unit* und *description* einer Variable haben rein informativen Charakter. Der *timestamp* markiert, wann der Wert der Variable das letzte mal geändert wurde. Die

Source gibt an, welcher Agent die Variable manipuliert oder versendet hat. Mit der *shared* Markierung wird bestimmt, ob die Variable nur zwischen Modulen eines Agenten, oder auch an andere Agenten versendet werden soll. Mittels der *rdf_class* ist es möglich, zusätzliche Semantik zu einer Variablen hinzuzufügen. Hiermit kann eine Verbindung zu einer RDF-Klasse in einer externen Ontologie hergestellt werden, zum Beispiel kann eine Variable als `brick:Temperature_Sensor` aus der Brick-Ontology markiert werde.

Der *DataBroker* ist für die Verwaltung und Ausführung von sogenannten *Callbacks* verantwortlich. Ein Callback ist eine Funktion, die ein Modul auf eine bestimmte Variable registriert. Die Registrierung ist spezifisch im Bezug auf *Alias* und *Source* der Variable. Wird nun eine Änderung dieser Variable im *DataBroker* registriert, so wird diese Funktion ausgeführt, mit der neuen Variable als Parameter. So müssen Module nicht zyklisch Variablen abfragen, sondern werden sofort benachrichtigt und können asynchron und reaktiv handeln.

Environment

Jeder Agent ist in einem *Environment* eingebettet. Die Environment hat dabei zur Aufgabe, die zeitlichen Abläufe innerhalb eines Rechenprozesses zu koordinieren. Mehrere Agenten können sich dasselbe *Environment* teilen, sofern sie im selben Rechenprozess sind. Ein funktionales Modul kann im *Environment* einen oder mehrere *Process*'es registrieren, die zyklisch eine Aufgabe durchführen, wie beispielsweise das Versenden von Messdaten, oder die Berechnung von Stellgrößen in einem Reglermodul mit fester Abtastzeit. Das *Environment* koordiniert die Ausführung dieser Prozesse innerhalb eines Rechenprozesses und hilft so, dass innerhalb eines *Environment*s keine Race-Conditions auftreten. Für das Environment können verschiedene Konfigurationen gewählt werden. So kann entschieden werden, ob das Multiagentensystem als Simulation (also so schnell wie möglich), in Echtzeit oder in skalierter Echtzeit ausgeführt werden soll. Ein Multiagentensystem, das über mehrere Rechenprozesse verteilt ist, also auch mehrere Environments besitzt, muss immer in Echtzeit oder skalierter Echtzeit ausgeführt werden, damit die Prozesse der Agenten in verschiedenen *Environment*s nicht auseinander laufen. Die Implementierung des *Environment* ist eine Erweiterung des SimPy Environment SimPy-Team.

Module

Funktionale Module können auf verschiedene Arten in die *AgentLib* eingebunden werden. Module sind eine Kombination aus einer Pydantic-basierten (`https://docs.pydantic.dev/`) Konfigurationsklasse und einer Modulklasse, die von der *BaseModule*-Klasse erbt. In der Konfiguration können Parameter des Moduls, sowie Agentenvariablen (die Kommunikationsobjekte der *AgentLib*) definiert werden. In der Modulklasse selbst, sind die Funktionen des Moduls definiert. So können Prozesse im *Environment* und Callbacks im *DataBroker* registriert werden.

1. Standardmodule sind im Kern der Bibliothek enthalten. Dazu zählen unter anderem verschiedene Kommunikatoren, ein Simulationsmodul, und Basisregler wie PID-Regler oder Hysteresregler. Diese Module haben einen Namen und können einfach durch Spezifikation des Namens in der Agentenkonfiguration verwendet werden.
2. Funktionale Module können in Plugin-Bibliotheken definiert werden. Nach Installation des Plugins können diese Module ebenfalls über ihren Namen eingebunden werden. Im Rahmen des Projekts AGENT wurde ein Plugin geschrieben, welches Module für die (verteilte) modellprädiktive Regelung definiert. Im Projekt „EnOB: DZWi “ (FKZ: 03EN1022B)) wurde die *AgentLib* ebenfalls genutzt, und um ein Plugin für digitale Zwillinge erweitert.
3. Individuelle Module können als Python-Dateien direkt über ihren Pfad eingebunden werden, sofern der Pfad in der Umgebung, in der der Agent ausgeführt wird, verfügbar ist.

Im Folgenden werden zuerst einige Standardmodule vorgestellt, sowie Module zur Anbindung an die Feldebene. Dazu gehören die Kommunikatormodule, das Simulatormodul, und die Module für die Anbindung über Fiware und Azure. In den Abschnitten 6.5 und 6.6 werden weitere Module vorgestellt, insbesondere für dezentrale und verteilte modellprädiktive Regelung.

Communicator-Modul Für die Kommunikation zwischen Agenten ist das Modul *Communicator* zuständig. Ein Agent benötigt immer mindestens ein Kommunikationsmodul, um mit seiner Umgebung zu interagieren. Diese Interaktion stellt eine Grundvoraussetzung für ein Agentensystem dar. In Abbildung 6.3 ist die Kommunikationsstruktur, in der die Kommunikatoren eingebunden sind, aufgezeigt. Hier ist beispielhaft die Kommunikation von einer Nachricht gezeigt, die von Agent 1 gesendet wird, und in einem funktionalen Modul von Agent 2 verarbeitet wird. Das *Communicator*-Modul von Agent 1 codiert eine Nachricht im `.json`-Format und versendet sie an einen Broker. Der Broker leitet die Nachricht weiter and das *Communicator*-Modul von Agent 2. Dieses decodiert den `.json`-Payload und erzeugt eine *AgentVariable*, und schickt diese an den *DataBroker*. Über *Alias* und *Source*, die in der *AgentVariable* enthalten sind, weiß der *DataBroker*, welche Callbacks er ausführen muss, und an welche Module die Variable weitergeleitet werden soll. In einem beispielhaften funktionalen Modul des zweiten Agenten ist die Variable registriert und wird somit an dieses Modul weitergeleitet. Dort könnte nun eine neue *AgentVariable* erzeugt werden, die über den *DataBroker* an den *Communicator* geleitet wird. *Communicator* hören prinzipiell auf alle Variablen in allen Modulen eines Agenten. Über das *shared*-Feld in der *AgentVariable* weiß der *Communicator*, ob die Variable verschickt werden soll. Ist das Feld aktiv, wird die Variable über den Broker an andere Agenten verschickt, und kann von deren Kommunikatoren empfangen werden.

Durch den Austausch von Kommunikatormodulen und ihrem Broker, kann das Agentensystem einfach von Testbetrieb auf Realbetrieb umgestellt werden, und neue Kommunikationsprotokolle können hinzugefügt werden, indem ein passendes *Communicator*-Module geschrieben wird. Im Kern der *AgentLib* sind bereits einige verschiedene Module enthalten. Dazu werden separate Klassen

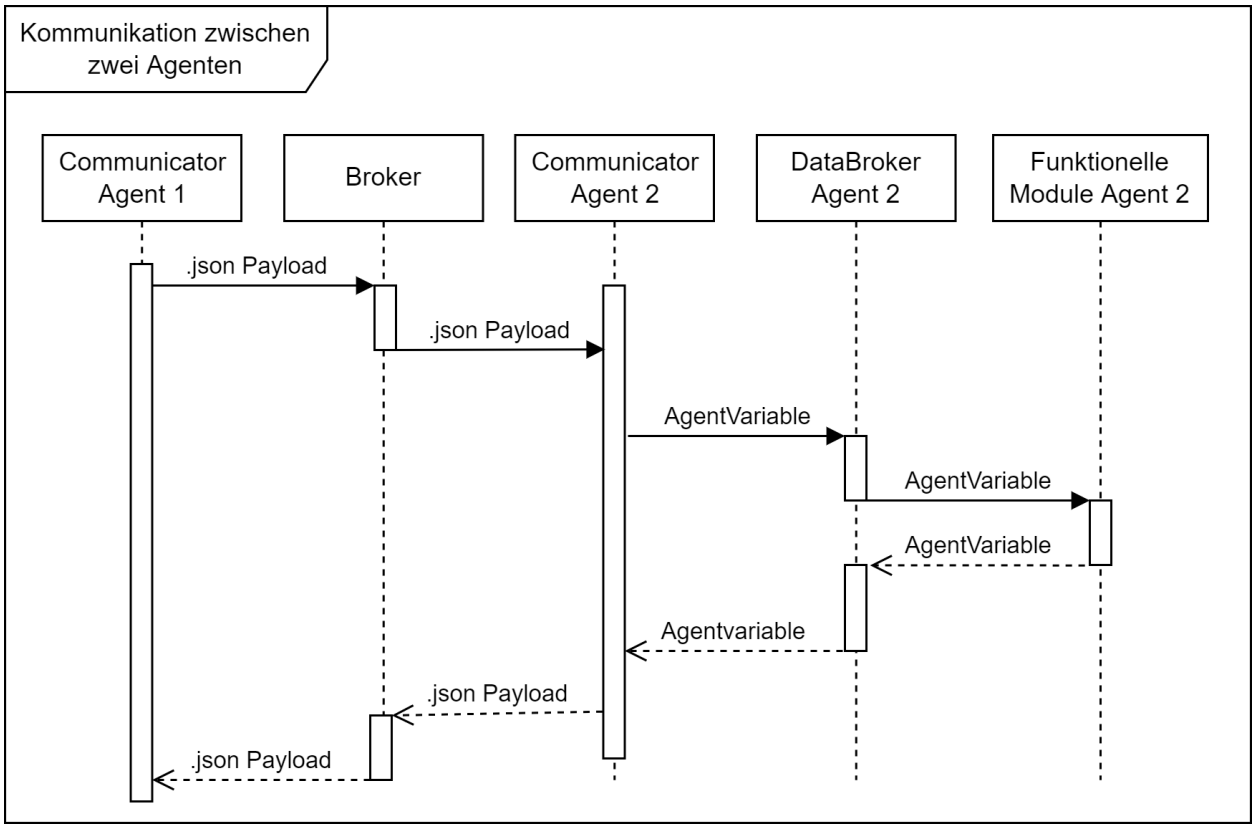


Abbildung 6.3: Struktur eines modularen Software-Agenten der AgentLib

erstellt, welche von der abstrakten Oberklasse `Communicator` erben. Zum Projektabschluss befinden sich folgende Kommunikatoren im Kern der Bibliothek:

- *local*
- *local_broadcast*
- *local_multiprocessing_broadcast*
- *mqtt*
- *clonemap*

Die lokalen Kommunikatoren *local* und *local_broadcast* stellen dabei die Kommunikation über einen Broker dar, auf den alle Agenten des Agentensystems Zugriff haben. Die Klasse `LocalBroadcastClient` ermöglicht die Kommunikation über einen lokalen Broadcast Broker. Dieser leitet Nachrichten an alle Agenten außer den Sender weiter. Für dieses Kommunikationsszenario müssen jedoch alle Agenten innerhalb eines einzelnen Rechnerprozesses ausgeführt werden. Die Kommunikation über den lokalen Broadcast Broker dient daher hauptsächlich der lokalen Entwicklung und dem Testen.

Die Klasse `MQTTClient` ermöglicht die Kommunikation über MQTT. Für die Implementierung wird die PahoMQTT Bibliothek verwendet. Die Kommunikation über MQTT bildet bereits eine Kommunikationsstruktur ab, wie sie auch für IoT-Systeme typisch ist. Das MQTT Protokoll ist

ein leichtgewichtiges Nachrichtentransportprotokoll, welches sich hoher Beliebtheit im Rahmen von IoT-Anwendungen erfreut. Der `MQTTClient` wird mit der URL für den MQTT-Broker konfiguriert. Somit können die Agenten auf verschiedenen Rechnerprozessen ausgeführt werden, sofern sie über eine Netzwerkanbindung verfügen.

Der *MultiprocessingClient* nutzt das *multiprocessing* Modul der Python-Standardbibliothek, um Agenten, die in verschiedenen Prozessen gestartet wurden miteinander zu verbinden. So können die Eigenschaften einer parallelen Simulation mit echten Verzögerungen getestet werden, ohne dass Zugriff auf einen MQTT Broker benötigt wird.

Weiterhin wurde ein Communicator für cloneMAP entwickelt und in die *AgentLib* integriert. Die entsprechende Communicator Klasse heißt `CloneMAPClient`. Dieser Communicator bindet die Python Bibliothek clonemapy ein, welche die Entwicklung von cloneMAP-kompatiblen Agenten in Python ermöglicht. Die clonemapy Bibliothek bietet eine Klasse `Agent` an, welche sämtliche Funktionalität für die Agentenkommunikation und Interaktion mit anderen cloneMAP Modulen implementiert. Eigene Agenten können durch Vererbung von dieser Klasse implementiert werden. Daher wird in der *AgentLib* eine Klasse `CloneMAPAgent` erstellt, welche von der *Agent* Klasse aus der cmapy Bibliothek erbt. In der Klasse `CloneMAPAgent` wird die Konfiguration des Agenten ausgelesen. Anschließend wird ein Objekt der `Agent` Klasse aus der *AgentLib* erstellt. Dem Konstruktor wird die zuvor ausgelesene Konfiguration übergeben. Der Agent erstellt ein Communicator Objekt. Dazu wird die neu erstellte Klasse `CloneMAPClient` genutzt. Diese erhält das Objekt der `CloneMAPAgent` Klasse und greift darüber auf die Methoden für Kommunikation zu, welche von clonemapy implementiert werden. cloneMAP ermöglicht peer-to-peer Kommunikation und Kommunikation über MQTT. Im Rahmen dieses Projekts wird nur die Kommunikation via MQTT genutzt.

Simulator-Modul Im Rahmen des Arbeitspaketes 3 wurde ein Modul entwickelt, welches die direkte Kopplung eines Simulationsmodels mit einem oder mehreren zu testenden Regelungsagenten ermöglicht. Dieser Softwarebaustein wurde in die *AgentLib* als *Simulator* integriert.

Der *Simulator* erlaubt es, verschiedene Modelle zu simulieren. Dabei hört der *Simulator* automatisch auf Eingänge des Modells und aktualisiert diese, wenn von anderen Agenten neue Werte, beispielsweise Stellwerte oder Messungen von Störgrößen verschickt werden. Im Gegenzug sendet der *Simulator* in regelmäßigen Abständen die Werte seiner Ausgänge. Somit bietet sich der *Simulator* dazu an, in der Entwicklungsphase der Regelalgorithmen das echte Gebäude zu ersetzen.

Im Laufe des Vorhabens wurden verschiedene Modelltypen integriert:

- Lineare Modelle
- Nichtlineare Modelle (CasADi)
- FMU-Modelle

Als einfachsten Modelltyp seien hier lineare Modelle genannt. Diese können einfach als System im Zustandsraum definiert und schrittweise simuliert werden. Um nichtlineare Modelle zu definieren,

ist eine symbolische Sprache notwendig, um die differenziellen und algebraischen Gleichungen zu definieren. Im Laufe des Vorhabens viel hierbei die Wahl auf das Python-Paket CasADi [Andersson et al., 2019], welches sich durch Rechenzeiten und eine einfache Integration der Modelle in modellprädiktive Regler auszeichnet. Mit der Integration von FMUs (Functional Mockup Unit) lassen sich schlussendlich auch komplexe Modelle, die beispielsweise in Modelica oder Simulink definiert wurden, in die *AgentLib* einbinden. Ein weiterer Vorteil von FMUs ist, dass sie auf dem Industriestandard FMI basieren, und die Entwicklung eines FMU-Modells unabhängig vom Agentensystem ist. Da für die Integration eines FMU-Modells in die *AgentLib* nur eine einzige Datei in der Umgebung des Agenten verfügbar sein muss, können so vielfältige und komplexe Modelle auf eine einfache Art und Weise ins Agentensystem eingebunden werden.

Basisregler-Module In der *AgentLib* sind zwei Basisregler integriert - ein PID-Regler und ein Hystereseregler. Die beiden Regler arbeiten mit Callbacks, sodass ein neuer Stellwert sofort als Reaktion auf einen erhaltenen Messwert gesendet wird. Im folgenden wird ein Beispielsystem gezeigt, dass diese Regler - sowie die anderen Kernmodule der *AgentLib* demonstriert. Dabei wurde ein Simulationsmodell eines Raums in Modelica erstellt und als FMU exportiert. Es wurden zwei Simulationsagenten für zwei Räume aufgesetzt und jeweils ein Agent mit Hystereseregelung und einer mit PID-Regelung. Die Regler bestimmen jeweils die notwendige Heizleistung, um die Temperatur auf einem Zielwert zu halten. In Auflistung 6.2 ist der notwendige Code zur Ausführung des Multiagentensystems zu sehen. In Auflistung 6.3 ist die `.json`-Konfiguration für den Hystereseregler zu sehen, und Auflistung 6.4 zeigt die notwendige Konfiguration des Simulatoragenten.

Auflistung 6.2: Multiagentensystem Beispiel

```
    env_config = {"rt": False}
    mas = LocalMASAgency(env=env_config,
                         agent_configs=[
                             "configs/TRYSensor.json",
                             "configs/Room2.json",
                             "configs/Room1.json",
                             "configs/PIDAgent.json",
                             "configs/BangBangAgent.json"
                         ],
)
```

Auflistung 6.3: Hystereseregler

```
{
  "id": "BangBangAgent",
  "modules": [
    {
      "module_id": "myBangBang",
      "type": "BangBang",
```

```
      "ub": 294.15,
      "lb": 292.15,
      "gain": 1000,
      "input": {
          "name": "u",
          "alias": "T_air",
          "value": 293.15
        },
      "output":
        {
          "name": "y",
          "alias": "Q_flow_heat",
          "shared": true
        }
    },
    {
      "module_id": "ComLocal",
      "type": "local",
      "subscriptions": ["Room2Agent"]
    }
  ]
}
```

Auflistung 6.4: Definition einer Variablen

```
{
  "id": "Room1Agent",
  "modules": [
    {
      "module_id": "room",
      "type": "simulator",
      "model": {
        "type": "fmu",
        "path": "models/SimpleRoom.fmu"
      },
      "measurement_uncertainty": 0.0001,
      "t_sample": 50,
      "save_results": true,
      "inputs": [
        {
          "name": "Q_flow_heat",
          "value": 0
        },
        {
          "name": "T_oda",
          "value": 273.15
        }],
      "outputs": [
```

```
        {
          "name": "T_air"
        }
      ]
    },
    {
      "module_id": "ComLocal",
      "type": "local",
      "subscriptions": ["PIDAgent", "TRYSensorAgent"]
    }
  ]
}
```

In Abbildung 6.4 sind die Simulationsergebnisse zu sehen. Wie erwartet, liefert der PID-Regler eine bessere Regelqualität.

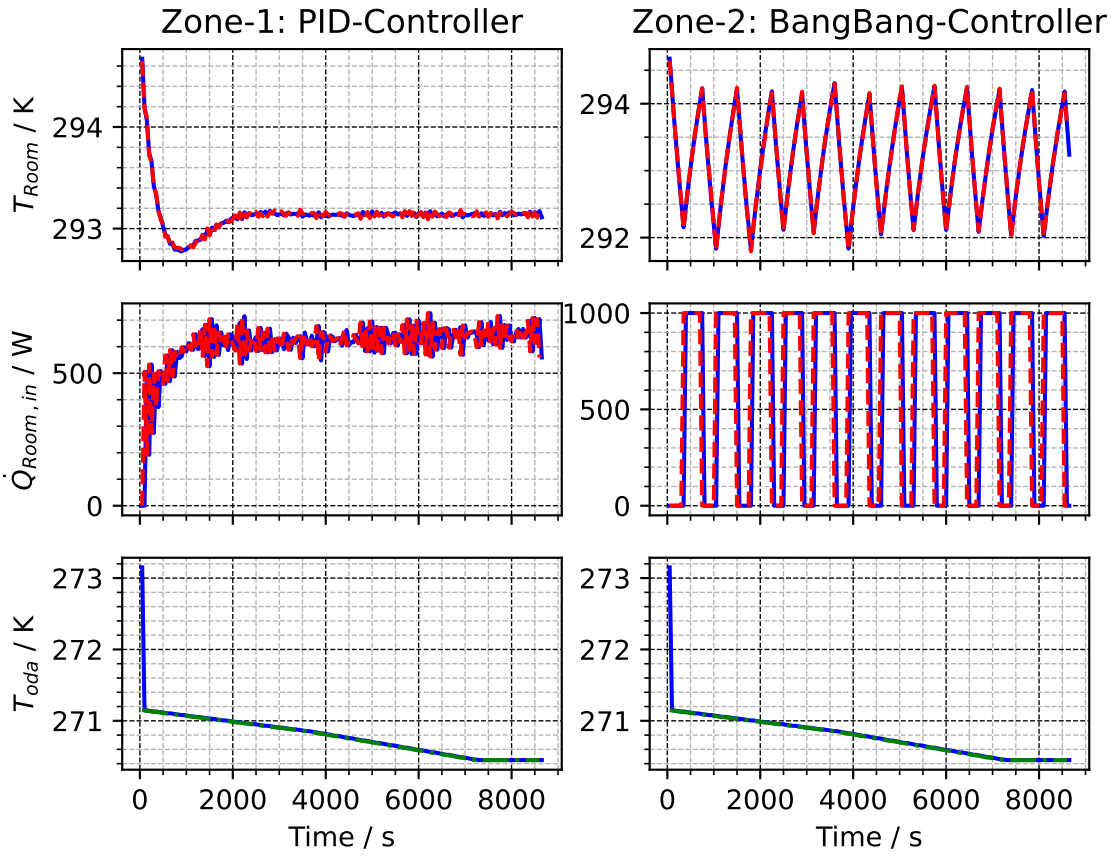


Abbildung 6.4: Simulationsergebnisse des “Room-MAS” Beispielsystems.

6.3.2 CloneMAP

Für die Entwicklung des Agentensystems wird die Plattform cloneMAP (cloud-native Multi-Agent Platform) verwendet. Diese wurde von der RWTH im Rahmen des BMBF Projekts ENSURE entwickelt. Der Fokus bei der Entwicklung lag auf einer hohen Skalierbarkeit sowie Ausfallsicherheit der Plattform.

CloneMAP und andere Multi-Agenten-Plattformen

Die Foundation for Intelligent Physical Agents (FIPA) stellt eine Standardisierung für Multi-Agent-Plattformen (MAPs) bereit. Das Zusammenspiel von Agent Management System (AMS), IoT Agents und optionalem Directory Facilitator (DF) sind hier definiert [Dähling et al., 2021].

Eine der bekanntesten MAPs ist das Java Agent DEvelopement Framework (JADE), das eine FIPA-konforme Middleware nutzt [Bellifemine et al., 2007; Kravari and Bassiliades, 2015a]. Andere Beispiele sind AgentScape, MadKit oder Magentix [Dähling et al., 2021]. Ein Vergleich von acht verschiedenen MAPs basierend auf unterschiedlichen Kriterien hat ergeben, dass JADE unter diesen über die beste Kommunikationsperformance verfügt [Leitão et al., 2016]. Zum gleichen Ergebnis kam eine Gegenüberstellung der drei Plattformen JADE, AgentScape und MadKit - allerdings mit der Erkenntnis, dass sowohl JADE als auch AgentScape durch ihre zentralisierte DF-Umsetzung gegenüber MadKit an Skalierbarkeit einbüßen [Alberola et al., 2010].

Das Thema Skalierbarkeit war auch für die Entwicklung von Magentix relevant: diese MAP ist verteilbar auf verschiedene Maschinen, indem jeder Agent als eigener OS Thread läuft. Daraus resultiert zwar (bis zu bestimmtem Agentenvolumen) im Vergleich zu JADE eine erhöhte Performance in der Kommunikation, allerdings führt die Einführung eines eigenen Threads für jeden einzelnen Agenten zu erheblichem Overhead, und könnte je nach Anzahl und den verfügbaren Ressourcen im Betriebssystem zu Engpässen führen [Alberola et al., 2013; Dähling et al., 2021].

Die Fehlertoleranz der bisher angesprochenen MAPs bewegt sich auf der Ebene einzelner Agenten. Wenn ein Agent ausfällt, kann ein zugehöriges MAS reagieren und diesen z.B. automatisch neu starten. Im Fall von Störungen in der Plattform selbst greift dies nicht. Im Gegensatz zu diesen vorgestellten Systemen macht die MAP SAGE dies durch ihre verteilte Architektur möglich. Bezüglich der Performance wurde gezeigt, dass SAGE für bis zu 400 Agenten gut doppelt so schnell ist wie JADE. Allerdings ist SAGE anfällig für Fehler und Ausfälle auf Agenten-Ebene, da diese nicht vom System selbst bewältigt werden [Ahmad et al., 2005].

Ein JADE-basiertes Multi-Agent-System (MAS) kann über mehrere Hosts verteilt aufgebaut werden [Bellifemine et al., 2007]. Das Elastic-JADE Konzept geht einen Schritt weiter und überträgt das verteilte System in die Cloud [Siddiqui et al., 2012]. Dies löst zwar direkt Hardware-bezogene Probleme, allerdings nicht die Hürden für Skalierbarkeit, die direkt mit der JADE-Architektur zu tun haben (z.B. den Bottleneck durch einen zentralen DF).

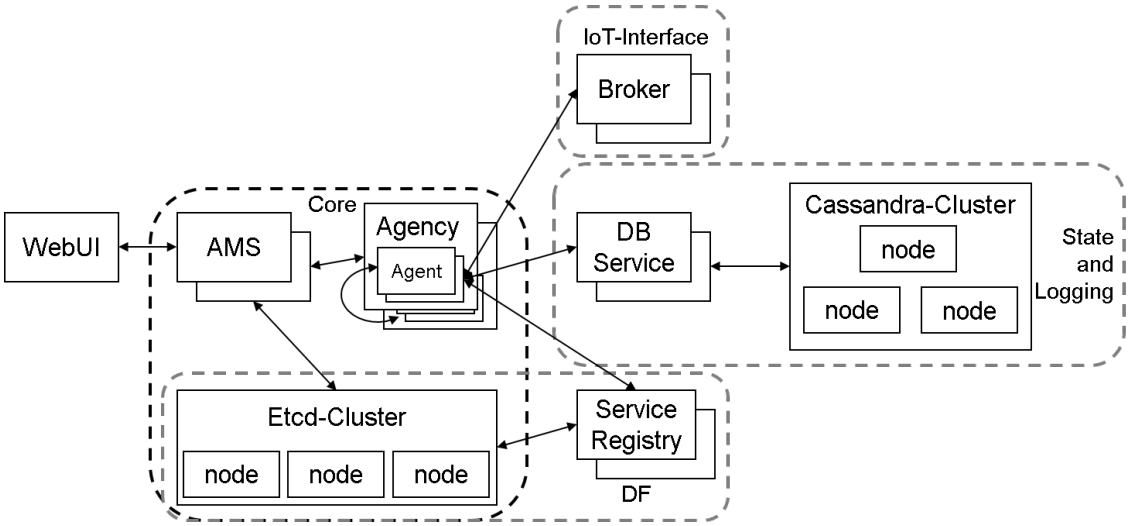


Abbildung 6.5: cloneMAP Architektur

Multi-Agenten-Systeme für IoT-Anwendungen erfordern besonders hohe Skalierbarkeit. Aktuelle Studien zu diesem Thema verwenden vornehmlich JADE, und führen insgesamt zu der Erkenntnis, dass die zuvor genannten Bottlenecks und Einschränkungen seine Nutzbarkeit im IoT-Bereich erheblich beeinträchtigen [Leitão et al., 2016; Dähling et al., 2021].

Für die Verwendung in AGENT liegen die relevanten Engpässe der genannten MAPs vor allem bei Fragen der Skalierbarkeit und Fehlertoleranz. Zwar gehen einige dieser Plattformen diese Herausforderungen durch verteilte Implementierungsansätze an, sind aber dennoch nicht auf sehr große Zahlen von Agenten ausgelegt.

CloneMAP hebt sich von den genannten Systemen ab, indem es MAS mit Container-Orchestrierung und Cloud Computing zusammenführt. Dies macht erhebliche Verbesserungen in der Skalierbarkeit möglich. Zudem wird sowohl Fokus auf Fehlertoleranz auf Plattform-Ebene gelegt, als auch Recovery auf Agenten-Ebene unterstützt [Dähling et al., 2021].

CloneMAP Aufbau und Einbindung im Projekt

cloneMAP ist aus vier Modulen aufgebaut, die gemäß dem Microservice Ansatz unabhängig voneinander verwendet werden können. Die gesamte Architektur von cloneMAP ist in Abbildung 6.5 dargestellt. Im Rahmen des Projektes AGENT wird lediglich das *Core* Modul verwendet, welches aus den beiden Teilen AMS (agent management system) und Agency besteht. Das AMS (agent management system) hat die Aufgabe Agenten zu starten und ihren Status zu überwachen. Um den Konfigurationsaufwand für die Inbetriebnahme des Agentensystems so gering wie möglich zu halten, soll für das Starten der Agenten ein Plug & Play Mechanismus entwickelt werden. Das bedeutet, dass die Registrierung eines IoT Gerätes in FIWARE automatisch den Start eines entsprechenden Agenten auslöst. Dazu wird das AMS mit FIWARE gekoppelt, sodass es eine Benachrichtigung über die Registrierung von Geräten erhält. Anschließend kann dann der Agent gestartet werden. Dabei

soll es auch möglich sein unterschiedliche Agenten für unterschiedliche Gerätetypen auszuführen. Das Submodul Agency hat folgende Eigenschaften.

- Agenten werden in Agency-Containern ausgeführt.
- Eine Agency enthält mindestens einen Agenten.
- Es können aber auch mehrere Agenten pro Agency ausgeführt werden.
- Die Agency erhält vom AMS die Konfiguration des zu startenden Agenten und führt diesen anschließend aus.
- Das Starten des Agency-Containers wird von Kubernetes durchgeführt.
- Die Agencies ermöglichen die Kommunikation von Agenten untereinander.

Die Interaktion von Agenten mit den zugehörigen Geräten erfolgt wiederum über FIWARE. Sensordaten der Geräte werden über die FIWARE IoT-Agents in der Orion DB abgelegt. Dort können sie über den Orion Contextbroker von den Agenten abgerufen werden. Diese ermitteln entsprechend der zu entwickelnden Algorithmen die Stellgrößen der Aktorik und schreiben diese zurück in die FIWARE Plattform. Von dort werden sie von den IoT-Agents an die IoT-Geräte weitergeleitet.

Die Gesamtarchitektur der Plattform bestehend aus cloneMAP und FIWARE ist in Abbildung 6.6 dargestellt.

Erweiterungen von CloneMAP im Projektzeitraum

Im Projektzeitraum wurde die Bibliothek clonemapy (zunächst unter dem namen cmapy) entwickelt, mit der die Implementierung von Agenten in Python möglich ist. Diese ist genau wie cloneMAP selber als open-source Projekt verfügbar [3]. Durch den modularen Aufbau von cloneMAP muss für die Nutzung einer anderen Programmiersprache lediglich die Agency, welche die Agenten ausführt, in die gewählte Sprache übersetzt werden, und nicht die gesamte Plattform. Dies ermöglicht theoretisch die Nutzung verschiedener Programmiersprachen. Die Projektpartner haben sich auf Python als Programmiersprache für die Agentenentwicklung geeignet.

Die Bibliothek clonemapy implementiert die Aufgaben der Agency für die Agentenausführung und die API für die Kommunikation mit anderen Modulen und zwischen Agenten. Eine Agency kann mehrere Agenten enthalten. In diesem Fall wird jeder Agent in einem eigenen Prozess ausgeführt. Dies ermöglicht die parallele Ausführung von Agenten auf mehreren CPUs.

Für die Kommunikation stehen mehrere Möglichkeiten zur Verfügung. Die direkte Kommunikation zwischen zwei Agenten (peer-to-peer) erfolgt über die cloneMAP-API, welche http nutzt. Darüberhinaus kann auch MQTT genutzt werden. Für diesen Zweck enthält jeder Agent einen MQTT Client. Auch eine Kommunikation zwischen Agenten, oder Agenten und IoT-Geräten, über die FIWARE-Plattform ist möglich.

[3] https://git.rwth-aachen.de/acs/public/cloud/mas/clonemapy

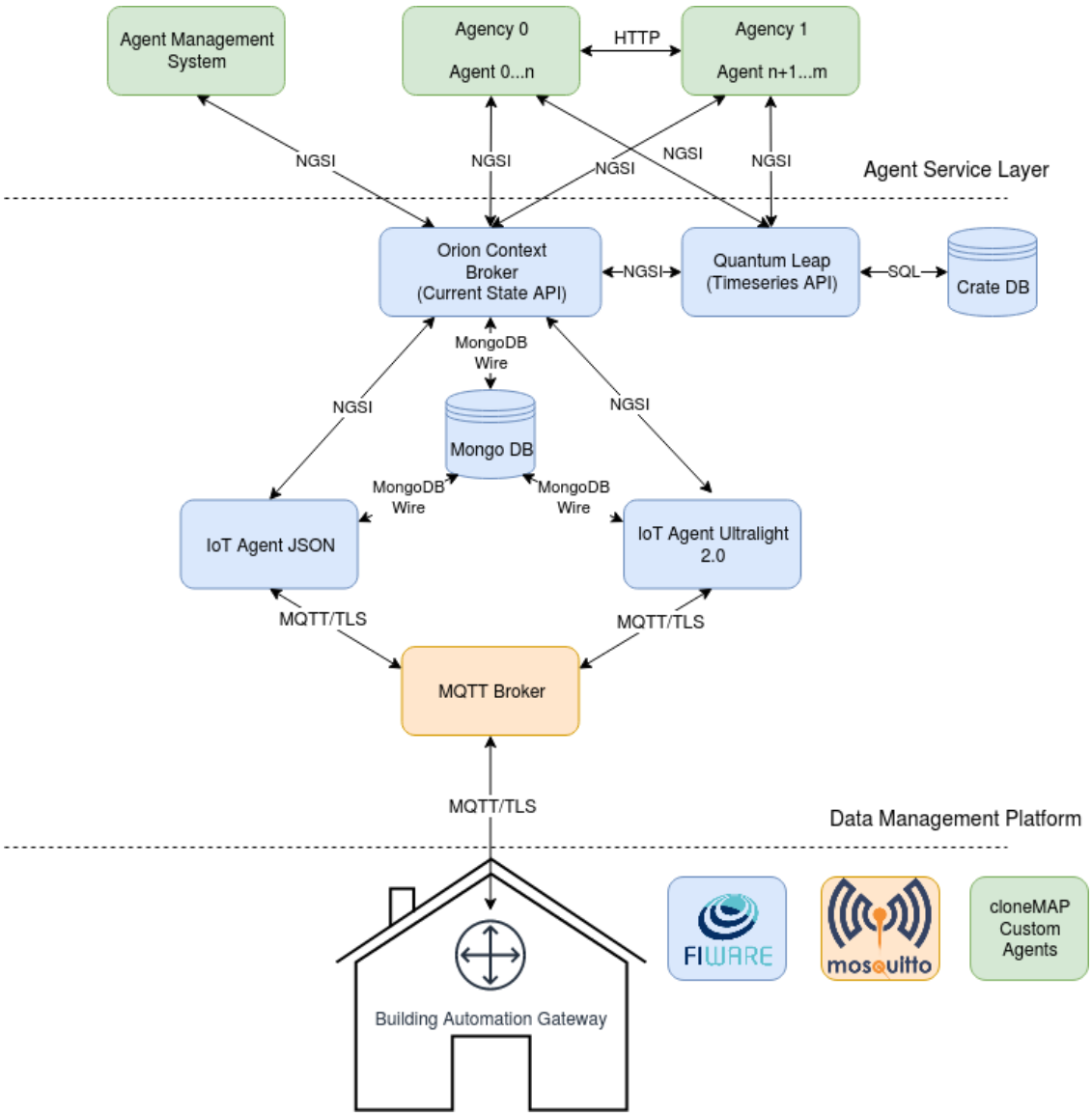


Abbildung 6.6: Gesamte Plattform bestehend aus cloneMAP und FIWARE

Weiterhin wurde die clonemapy Bibliothek in einen Communicator, wie in Abschnitt 6.3.1 beschrieben, eingebunden. Dadurch wird die komfortable Nutzung der Bibliothek, und somit die Implementierung von Agenten in cloneMAP, als Teil des entwickelten Agenten-Templates ermöglicht.

Im Laufe des Projekts wurden zudem einige Verbesserungen von cloneMAP in Bezug auf Nutzbarkeit umgesetzt. Zum einen wurde damit begonnen eine web-basierte Benutzeroberfläche zu implementieren. Diese ermöglicht die Steuerung von cloneMAP im Webbrowser über grafische Bedienelemente. Zuvor war die Bedienung von cloneMAP nur über Skripte oder eine Konsole möglich. Für die Implementierung werden HTML, CSS und Javascript eingesetzt. Die Web-Oberfläche ist, wie die gesamt Plattform, als open-source Projekt verfügbar.

Weiterhin wurde die Ausführung von cloneMAP auf einem lokalen Rechner ermöglicht. cloneMAP wurde ursprünglich für die Ausführung in einem Kubernetes-Cluster entwickelt. Für das Entwickeln und Testen von Agentenalgorithmen ist die Nutzung eines Kubernetes-Clusters jedoch nicht zweckmäßig. Eine lokale Ausführungsmöglichkeit für cloneMAP löst dieses Problem. Bei lokaler Ausführung werden die einzelnen Microservices von cloneMAP lokal als Docker-Container gestartet. Das Starten von Containern mit Agenten, welches normalerweise von Kubernetes erledigt wird, übernimmt ein spezieller Container, welcher auf den Docker-Daemon des Hosts zugreifen kann. Wichtige Kubernetes-Features, wie horizontale Skalierbarkeit oder Ausfallsicherheit, werden von der lokalen cloneMAP Variante nicht zur Verfügung gestellt. Somit ist die lokale Ausführung von cloneMAP nur für das Entwickeln und Testen von Agenten, nicht aber für den produktiven Einsatz sinnvoll. Ein in der Agentlib bestehendes Beispiel mit mehreren Modellen unter Verwendung von ADMM wurde um eine cloneMAP-Variante erweitert, um die Ausführung und Kommunikation von mehr als zwei Agentlib-Agenten in einem cloneMAP-AMS sicherzustellen.

Zudem wurde für verbesserte Nutzerfreundlichkeit das Logging von agentlib-Agenten, die via cloneMAP ausgeführt werden, einfacher zugänglich gemacht.

6.3.3 Azure Iot Hub- und ADT-Kommunikator

Zur Anbindung des realen Gebäudes wurden entsprechende Konnektoren für die *AgentLib* entwickelt, um Agenten mit einem IoT Hub in Azure zu verbinden.

Die Telemetrie des Gebäudes wird mittels MQTT an einen Azure IoT Hub gesendet, um in der Cloud verarbeitet zu werden. Zur Anmeldung der Agenten an diesem IoT Hub wird ein Azure IoT Hub Device Provisioning Service (DPS) verwendet. Dazu wurde in der *AgentLib* ein Konnektor implementiert, der sich dorthin verbindet und eine passende Kommunikationsschnittstelle bereitstellt. Dieser Konnektor erlaubt es, Daten von einem IoT Hub zu lesen und dorthin zu schreiben; insbesondere ist es damit möglich, Daten an das ebenfalls mit diesem IoT Hub verbundenen Gebäude zu senden.

Darauf aufbauend wurde ein ADT-Kommunikator implementiert, der sich zusätzlich mit einer Instanz von Azure Digital Twins (ADT) verbindet. Für das Demonstrationsgebäude existiert ein

digitaler Zwilling als ADT, der über eingehende Telemetrie automatisch aktualisiert wird und damit zu jedem Zeitpunkt den aktuellen Zustand des Gebäudes bezüglich Topologie und Messdaten widerspiegelt. Der ADT-Kommunikator bietet innerhalb der *AgentLib* eine Schnittstelle zu einer ADT-Instanz, um an diese Anfragen zu senden.

Über die dort hinterlegte semantische Beschreibung der innerhalb des Modells einer MPC benutzten Variablen können diese so vor jedem Optimierungsschritt automatisch vom digitalen Zwilling des Gebäudes abgefragt werden.

6.3.4 Fiware

Neben der Anbindung an den kommerziellen IoT-Hub von Azure wird die Anbindung an die quelloffene und kostenfreie IoT-Plattform FIWARE realisiert.

Ziel des Projektes ist im ersten Schritt die simulative Bewertung von agentenbasierter Optimierung über FIWARE. Dazu wurde für FIWARE eine Anbindung an die bestehenden Simulationsmodelle implementiert. Somit kann jedes Simulationsmodell der im Rahmen von AP02 genutzten und erweiterten Modellbibliothek als Emulator eines realen Systems an FIWARE angebunden werden. Zur Anbindung an FIWARE wurden zwei *AgentLib*-Module implementiert:

- *FIWAREIoTAMQTTClient*
- *ContextBrokerCommunicator*

Der *FIWAREIoTAMQTTClient* übernimmt die Kommunikation des realen Systems bzw. des Emulators mit dem IoT-Agent aus FIWARE durch Nutzung eines MQTT-Brokers. Messwerte des Systems werden als *attribute* an den Context-Broker in FIWARE gesendet. Stellsignale an das System aus der Cloud werden als *command* von dem Context-Broker empfangen. Die korrekte Applikation eines empfangenen Stellsignals wird anschließend an den Context-Broker übermittelt. Somit kann ein Agent prüfen, ob gesendete Werte vom realen System übernommen werden. Dieser Ablauf ist im unteren Teil von Abbildung 6.7 dargestellt.

Der *ContextBrokerCommunicator* abonniert im Context-Broker jedwede Änderung in den Entitäten, welche überwacht werden sollen. Bei einer Änderung des Messwerts erhält das *ContextBrokerCommunicator*-Modul eine Nachricht über einen MQTT-Broker. Der Wert des *attribute* wird anschließend im Agenten an weitere Module über den *DataBroker* gesendet. Umgekehrt können Agentenmodule, z.B. ein ADMM-Modul, ein Stellsignal mit korrektem *alias* an den *ContextBrokerCommunicator* senden, welcher ein *command* an die Feldebene sendet. Dieser Ablauf ist im oberen Teil von Abbildung 6.7 dargestellt.

Zur vereinfachten Anbindung der entwickelten Module wurde weiterhin eine sogenannte *DeviceFactory* entwickelt. Für ein angegebenes Simulationsmodell werden automatisiert Geräte (Devices) in FIWARE angelegt. Ebenso werden die benötigten Konfigurationen der Module *ContextBrokerCommunicator* und *FIWAREIoTAMQTTClient* erzeugt. Somit können Nutzende ohne tief reichende Kenntnis über FIWARE Simulationen mit einer realen IoT-Plattform durchführen.

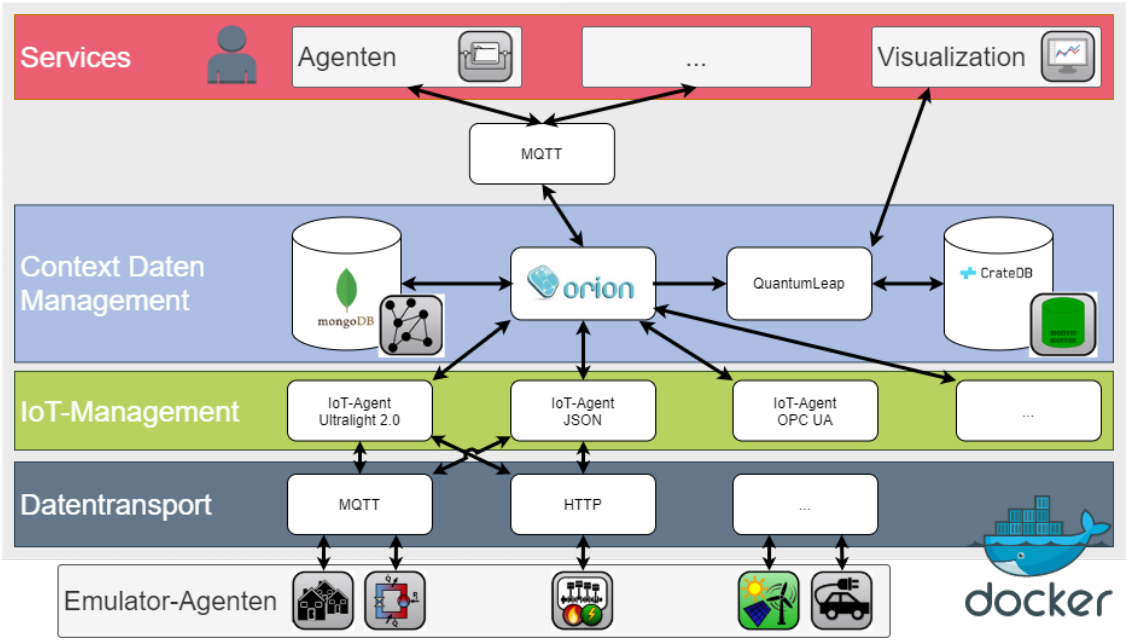


Abbildung 6.7: Anbindung von Emulator-Agenten und Regelungs-Agenten als Services über die FIWARE-Plattform

Zuletzt sei eine aktuelle Hürde bei der Simulation über FIWARE erwähnt. Durch die Nutzung von *SimPy* (SimPy-Team) in AP 04 kann die *AgentLib* in Echtzeit, beschleunigter Echtzeit oder so schnell wie möglich simulieren. In der Kopplung zur realen IoT-Plattform wird der Zeitstempel der Messwerte jedoch von der IoT-Plattform vorgegeben. Entsprechend sind zeitabhängige Algorithmen nur dann über reale IoT-Plattformen in beschleunigter Echtzeit zu nutzen, wenn die Algorithmen nicht sensitiv gegenüber der Zeit sind. Beispielweise können PID-Regler aktuell nur in Echtzeit getestet werden, da über die Zeit integriert wird. Zur Lösung des Problems könnte das Modul hinsichtlich einer Option zur Nutzung des internen Zeitstempels erweitert werden. Da kein direkter Bedarf hierzu im Projekt entstanden ist, wurde diese Option nicht realisiert.

6.4 Entwicklung eines Software-in-the-Loop-Frameworks

Neben den Effizienzansprüchen und der Emissionseinsparung muss die Gebäudeautomation insbesondere zuverlässig den Nutzerkomfort sicherstellen. Oftmals werden Automationsprogramme individuell für ein bestimmtes Gebäude entwickelt. Ein wichtiger Schritt ist hierbei ein umfassender Test der Automationsfunktionen. Hierbei wird die Reaktion des Programms auf eine Sollwertvorgabe überprüft. Die Effizienz wird hierbei in der Regel nicht betrachtet und kann erst nach längerem Betrieb bewertet werden und das Programm vor Ort angepasst werden. Dieser Prozess ist nicht nur aufwendig und langwierig, sondern auch kostenintensiv. Darüber hinaus wurde die Komplexität der zugrunde liegenden Kommunikationsinfrastruktur, in der Forschung häufig beim Vergleich von Regelungskonzepten vernachlässigt und daher nicht in die Bewertung einbezogen. Um diese For-

schungslücke zu schließen, ist ein umfängliches Gesamttestverfahren erforderlich. Dies wird nicht nur die Entwicklung und Bewertung intelligenter Regelungskonzepte in einer Cloudumgebung unter Verwendung verfügbarer Werkzeuge für die Co-Simulation, sondern auch die nahtlose Migration von der Erprobung zur praktischen Umsetzung in Gebäuden unterstützen. Insbesondere Letzteres erfordert die frühzeitige Integration der zugrundeliegenden Netzinfrastruktur in das Testverfahren.

Ziel des entwickelten Software in the Loop (SiL)-Frameworks ist daher die Bereitstellung eines Softwarepakets, welches es ermöglicht, die im Projekt entwickelten Agenten sowie deren Integration in das Gesamtregelungskonzept bereits in einem frühen Entwicklungsstadium testen zu können. Dies führt dazu, dass die Kommunikationsinfrastruktur bei der späteren Integration in das reale Gebäude nur geringfügig rekonfiguriert werden muss, was die Anzahl eventueller Fehlerquellen stark reduziert. Hierdurch soll die spätere Einbindung eines Agentensystems in den realen technischen Kontext mit allen Überwachungs-, Steuerungs- und Regelfunktionen sichergestellt werden. Insgesamt wird durch die Nutzung eines solchen virtuellen Prüfstandes zukünftig die Entwicklung, Inbetriebnahme sowie die Wartung von Gebäudeautomationssystemen auch über die Projektlaufzeit hinaus kostengünstig ergänzt und unterstützt.

6.4.1 Konzeptionierung eines virtuellen Prüfstands zur Entwicklung und Bewertung von Regelungskonzepten

Im Rahmen des Projekts wurde die Architektur für einen virtuellen Prüfstand entwickelt. Die Simulation eines Testszenarios basiert im Wesentlichen auf der Funktionsweise der im Projekt entwickelten Agenten selbst, welche in Abschnitt 6.3 und Abschnitt 6.5 näher beschrieben werden. Daher wird an dieser Stelle nicht näher darauf eingegangen. Dieser Ansatz wurde aufgrund der hohen konzeptionellen Ähnlichkeit zwischen Agentensystemen und IoT-Anwendungen als besonders zielführend erachtet. Hierdurch werden die Konzepte des Cloud-Computings und IoT-Systeme, deren Bedeutung für Energiesysteme zukünftig zunehmen wird, mit berücksichtigt. Das Verhalten einzelner realer Hardwarkomponenten wird durch Emulations-Agenten übernommen, die das Anlagenverhalten über Simulationsmodelle nachahmen. Hierfür wird das Simulationsmodul (Simulator), der in Abschnitt 6.3 beschriebenen Softwarebibliothek, verwendet. Ein Agent kann hierbei eine oder mehrere Komponenten abbilden. Aus Sicht einer späteren Steuerungsapplikation unterscheiden sich die Ein- und Ausgänge der Anlagensimulation später nicht mehr von ihren realen Gegenstücken, wobei Eingänge als Aktoren und Ausgänge als Sensoren dargestellt werden.

Die Gesamtstruktur der Architektur ist in Abbildung 6.8 dargestellt. Die einzelnen Bausteine werden im Folgenden beschrieben. Im Rahmen des Projektes wurden nicht alle Komponenten vollwertig, sondern teilweise prototypisch implementiert. Dies diente insbesondere der Demonstration der grundsätzlichen Funktionalität des Gesamtkonzeptes. Eine vollständige Ausgestaltung aller Komponenten des hier vorgestellten Prüfstandes lag jedoch nicht im Fokus des Gesamtprojektes.

Frontend: Grafische Benutzeroberfläche (*engl.* graphical user interface (GUI)) zur Interaktion mit der Plattform. Sie bietet nicht nur die Möglichkeit, den Testaufbau zu konfigurieren, sondern

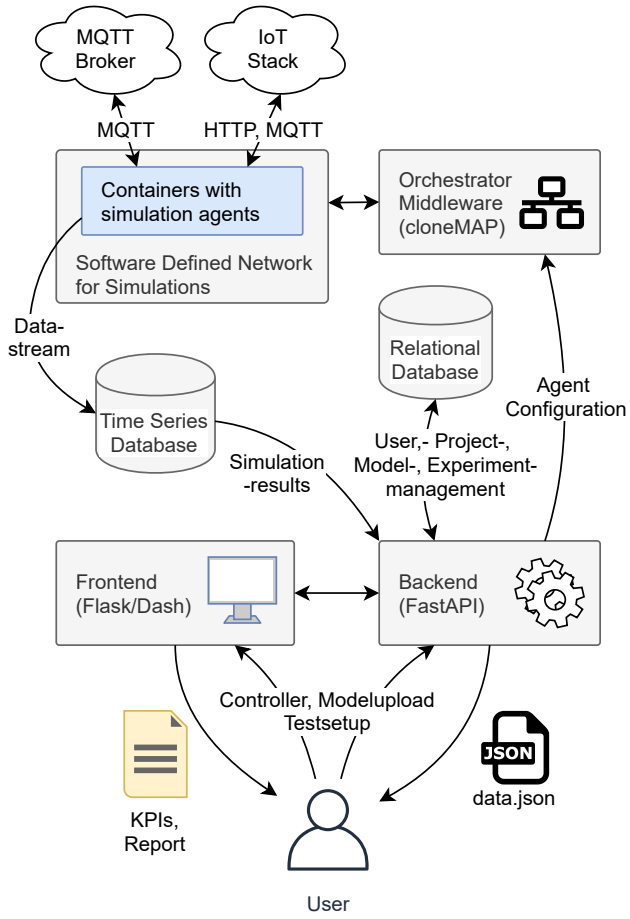


Abbildung 6.8: Architektur und Interaktion der *Microservice*-Infrastruktur die zusammen den virtuellen Prüfstand abbilden

auch den Upload und die Verwaltung von Simulationsmodellen, die innerhalb der Emulationsschicht verwendet werden sollen. Derzeit wird der Upload von functional mockup unit (FMU) und Modelica-Modellen unterstützt. Letztere werden jedoch innerhalb des Backends automatisch in FMUs konvertiert. Voraussetzung hierfür ist jedoch die vorherige Verfügbarmachung aller benötigter Modellbibliotheken sowie von Softwarelizenzen. Durch diese Modellverwaltung wird dem Nutzer ermöglicht, sehr flexibel seine Regelstreckenmodelle zu verändern oder zu erweitern. Virtuelle Experimente und Versuchsreihen können vom Nutzer im Rahmen von Projekten analog zu der Modellverwaltung angelegt werden.

Experimente ermöglichen es dem Nutzer unterschiedliche Simulationskonfigurationen zu untersuchen. Hierbei kann er nicht nur wie oben bereits erwähnt zwischen zwei Simulationsarten wählen, sondern auch die Verbindung zur IoT-Plattform, in der der Regelungsalgorithmus ausgeführt wird, und die auszutauschenden Variablen konfiguriert werden. Am Ende jedes Testdurchlaufs ermöglicht das Frontend die Visualisierung von Simulationsergebnissen. Um die einzelnen Durchläufe

und Konfigurationen vergleichbar zu machen, können hierbei die Ergebnisse mehrerer Durchläufe visuell überlagert werden. Im Rahmen des Projektes wurde eine GUI prototypisch als Webapp mit dem Python-Framewok Dash umgesetzt. Diese ist in den Abbildungen 6.9, 6.10 und 6.11 dargestellt.

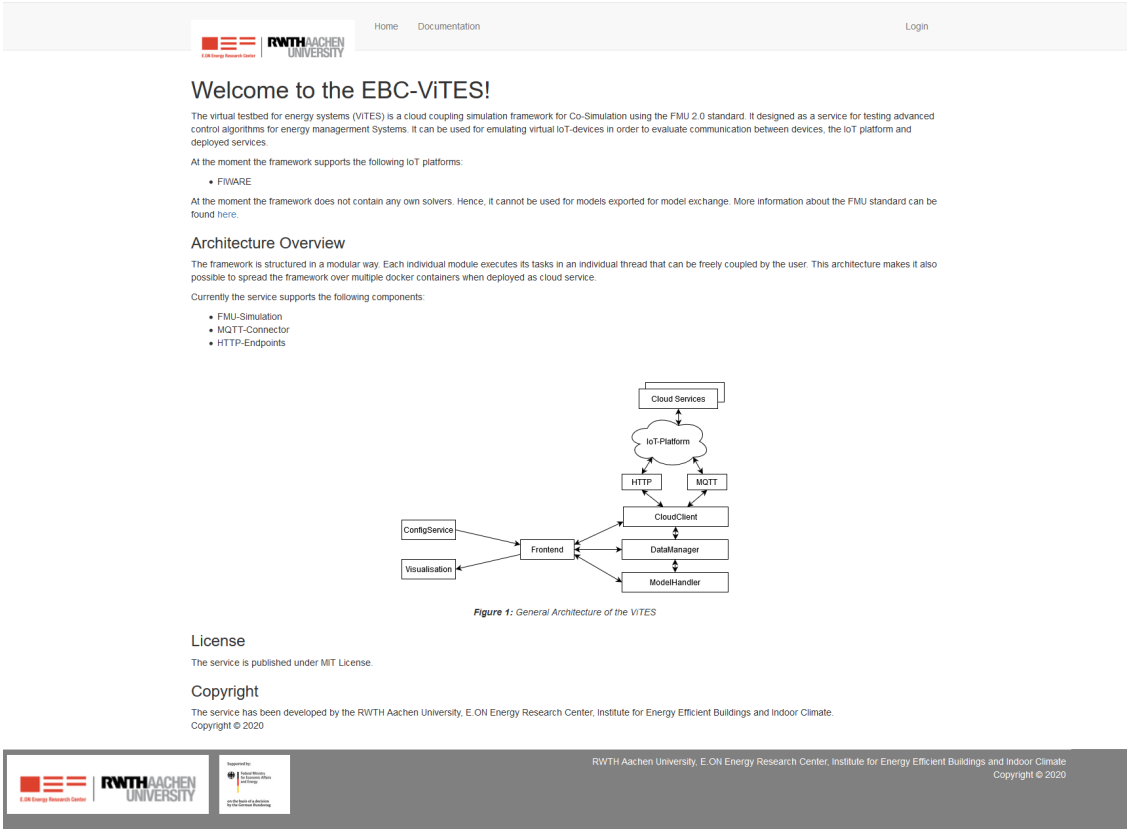


Abbildung 6.9: Webbasierte Bedienoberfläche des virtuellen Prüfstandes

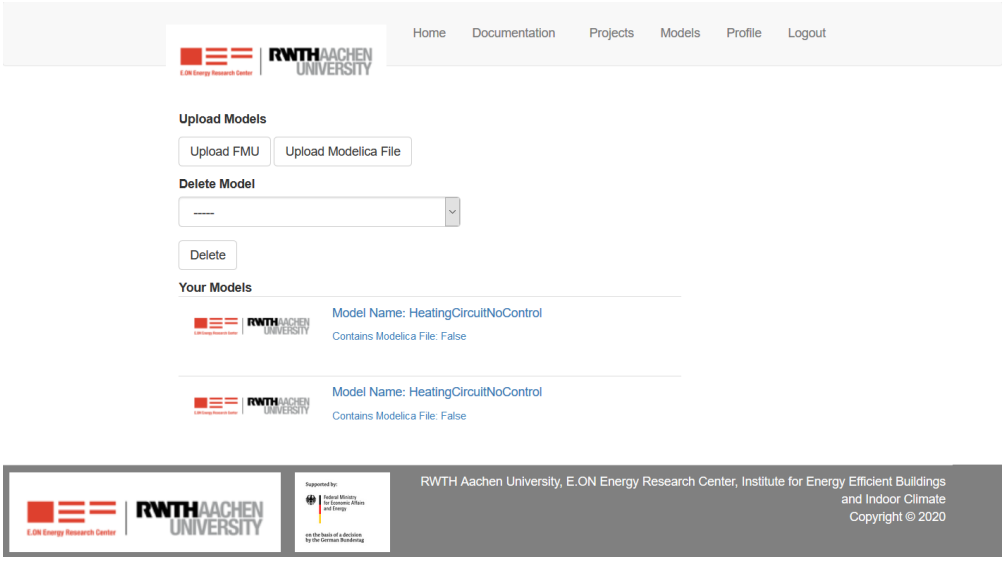


Abbildung 6.10: Modellverwaltung innerhalb der webbasierten Bedienoberfläche des virtuellen Prüfstandes

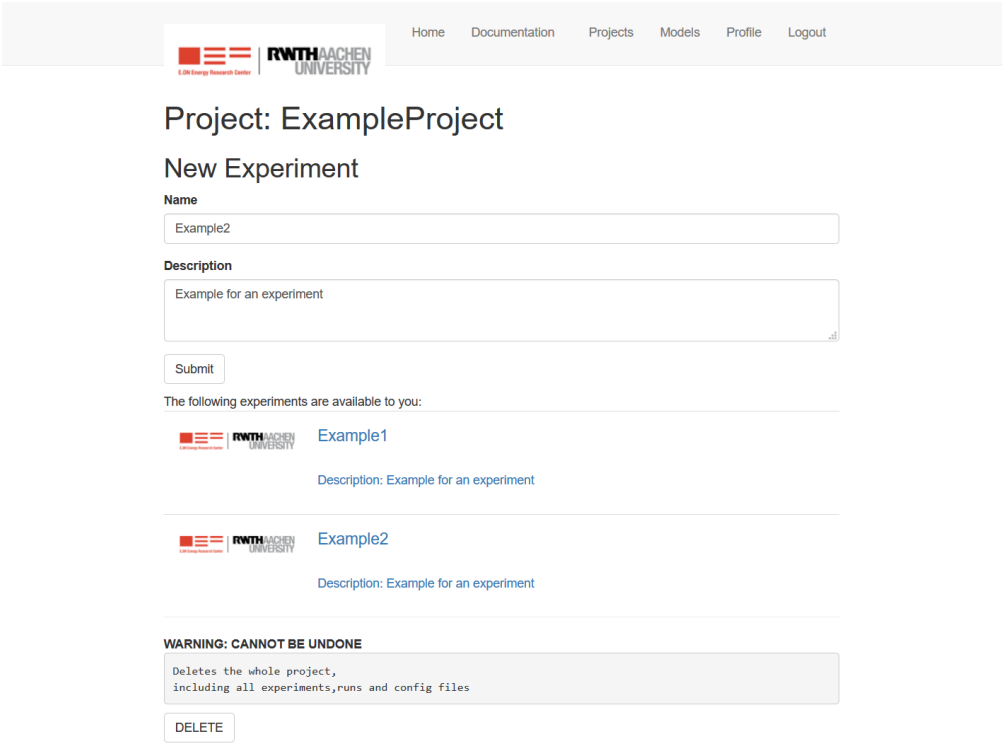


Abbildung 6.11: Experimentverwaltung innerhalb der webbasierten Bedienoberfläche des virtuellen Prüfstandes

Backend: Kern des virtuellen Prüfstandes. Es stellt alle Routinen zur Verwaltung von Benutzern, Projekten, Simulationsmodellen und Tests zur Verfügung. Darüber hinaus stellt es auch Funktionen für die Analyse von Simulationsergebnissen und die Berechnung von KPIs zur Verfügung. Schließlich

stellt es über seine RestAPI eine Schnittstelle für Software von Dritten zur Verfügung.

Relationale Datenbank: Persistente Datenablage, die für die Benutzer-, Projekt-, Modell- und Testverwaltung benötigt wird.

Zeitreihendatenbank: Effiziente Speicherung von Simulationsergebnissen und die Datenverarbeitung und Analyse im Backend.

Orchestrator-Middleware: Skalierung und Koordination von mehreren gleichzeitigen Simulationsszenarien. Diese Aufgabe wird von der im Projekt bereits vorgestellten cloudnative Multi-Agent Platform (cloneMAP) übernommen. cloneMAP übernimmt die Verwaltung der einzelnen Agenten und startet sie als Docker-Container innerhalb eines virtuellen Netzwerks. Damit wird die Nachahmung einer realen Netzwerkinfrastruktur erreicht. Durch die Verwendung von cloneMAP wird ebenfalls erreicht, dass der Prüfstand die in Abschnitt 6.3 beschriebene Bibliothek für eine agentbasierte Systemarchitektur, 1:1 nachbildet.

Message Queuing Telemetry Transport (MQTT)-Broker: Inter-Agenten-Kommunikation über ein für IoT-Anwendungen typisches Protokoll. Es wird entweder von Drittanbietern oder als zusätzlicher Microservice von der angekoppelten IoT-Plattform bereitgestellt. Agenten interagieren mit ihm über das bereitgestellte MQTT-Modul. Außerdem bietet es eine Schnittstelle zu Komponenten von Drittanbietern, die außerhalb des Frameworks ausgeführt werden. Die Nutzung als externer Kommunikationskanal erfordert unter Umständen die Einrichtung von State-of-the-Art-Sicherheitsmechanismen, während die interne Nutzung diese auch vernachlässigen kann.

IoT-Stack: Plattform, für die das spätere Agentensystem ausgelegt wird. Die Wahl des IoT-Stacks liegt in der Verantwortung des Anwenders. Aufgrund der Individualität der verschiedenen IoT-Stacks ist der spätere Benutzer jedoch auch dafür verantwortlich, die Routinen bereitzustellen, die der Agent benötigt, um mit dem Stack zu interagieren, d. h. plattformspezifische Kommunikationsmodule bereitzustellen. Innerhalb des Projektes wurde hier zunächst FIWARE als quelloffene Lösung eingesetzt. Die frühzeitige Einbindung des IoT-Stacks ermöglicht später einen reibungslosen Übergang von der Emulation zum realen Szenario.

6.4.2 Konzeptbewertung des stufenweisen Entwicklungsprozesses von Automationssystemen

Zum Nachweis des Konzepts des mehrstufigen Entwicklungverfahrens wurden mehrere Tests mit zwei Szenarien durchgeführt.

Um die rudimentären Anforderungen für die Konfiguration von Agentenmodulen und für die Kommunikation zwischen ihnen zu untersuchen wurde zunächst ein ereignisgesteuertes Ping-Pong-Experiment mit zwei Agenten durchgeführt. Außerdem erlaubt dieses Experiment den Test der Synchronisation mit einem globalen Zeitgeber, wenn ein Echtzeit-Setup verwendet wird. Das PingPong-Modul selbst ist als Modul der in Abschnitt 6.3 beschriebenen Softwarebibliothek geschrieben. Das Experiment ermöglicht es, sowohl eine lokale Kommunikation als auch eine Kommunikation über

MQTT mit einem öffentlich verfügbaren MQTT-Broker aufzubauen. Wie erwartet, zeigen beide Setups keine grundlegenden Unterschiede während der Laufzeit. Darüber hinaus funktioniert auch die Beschleunigung der Simulation in einem synchronisierten Setup. Wie erwartet, fügt die Kommunikation über MQTT jedoch eine kleine Verzögerung für jeden Kommunikationsschritt hinzu, weshalb sich die Gesamtausführungszeit erhöht. Ebenso erfordert der Verbindungsaufbau im MQTT-Setup einen erheblichen Zeitaufwand, der beim Simulationsaufbau oder bei der Entwicklung von Regelungskonzepten berücksichtigt werden muss.

In einem zweiten Szenario wurde das vorgestellte Framework für Untersuchungen einer verteilten Steuerung einer Vier-Raum-Wohnung angewandt. Jeder der Räume wurde nach dem High-Order-Ansatz aus der *AixLib* modelliert. Die Wärmeversorgung wurde durch Heizkörper realisiert. Während die Vorlauftemperatur direkt über eine Heizkurve eingestellt wird, steuert ein PID-Agent den Massenstrom über die Ventilöffnung der Heizkörper. Bei der Analyse wurde die IAE des Raumtemperaturreglers als KPI in Abhängigkeit verschiedener Kommunikationsschrittweiten analysiert. Alle Simulationen wurden synchron mit konstanten Modellparametern über den Temperaturverlauf eines Wintertages als Randbedingung ausgeführt. Diese Studie zielt auf die Identifizierung der minimal erforderlichen Schrittgröße ab, bei der die Regelungsqualität noch akzeptabel ist. Wir führten die Untersuchungen sowohl als lokale Simulation, als auch als verteilte Echtzeitsimulation durch. Zusätzlich wurde durch die Variation des Quality of Service (QoS) von 0 bis 2 der Einfluss der Kommunikation über MQTT betrachtet. Abbildung 6.12 stellt die Ergebnisse für einen Raum dar. Unabhängig von der untersuchten Realitätsnähe der Kommunikationsstruktur nimmt die Steuerungsqualität mit höheren Schrittweiten ab. Der lokale Kommunikator liefert jedoch für alle Abtastzeitpunkte die niedrigsten IAE-Werte. Dies ist eine direkte Folge der verteilten Co-Simulation. Die lokale Simulation wird erst fortgesetzt, wenn alle Nachrichten gesendet und empfangen wurden. Da der IAE bei einer QoS von 0 niedriger ist, konnte der Fehler auf die Latenzzeit beim Senden und Empfangen von Werten auf den Online-Broker zurückgeführt werden. Diese Latenzzeit wird in der Co-Simulation nicht berücksichtigt. Daher gehen die Informationen entweder verloren oder werden mit einer erheblichen Verzögerung empfangen. Alle Agenten wurden in einem Container ausgeführt. Da Python Multi-Thread-Aufgaben sequentiell ausführt, haben einige Variablen und Agenten eine implizit höhere Latenztoleranz. Daher war die Abnahme der Kontrollqualität je nach Agent und Variable stärker ausgeprägt. Um diese Erkenntnis zu berücksichtigen, wurde die Echtzeitausführung für Schrittgrößen größer-gleich 100 s mit einer expliziten Pause von 100 ms erneut durchgeführt Die Ergebnisse dieser Studie zeigen deutlich, dass die Berücksichtigung der Latenzzeit das Problem löst und zu den gleichen Ergebnissen wie die lokale Simulation führte. Man beachte auch, dass der IAE mit kleineren Schrittweiten gegen den IAE konvergiert, die man bei der gekoppelten Simulation erhält.

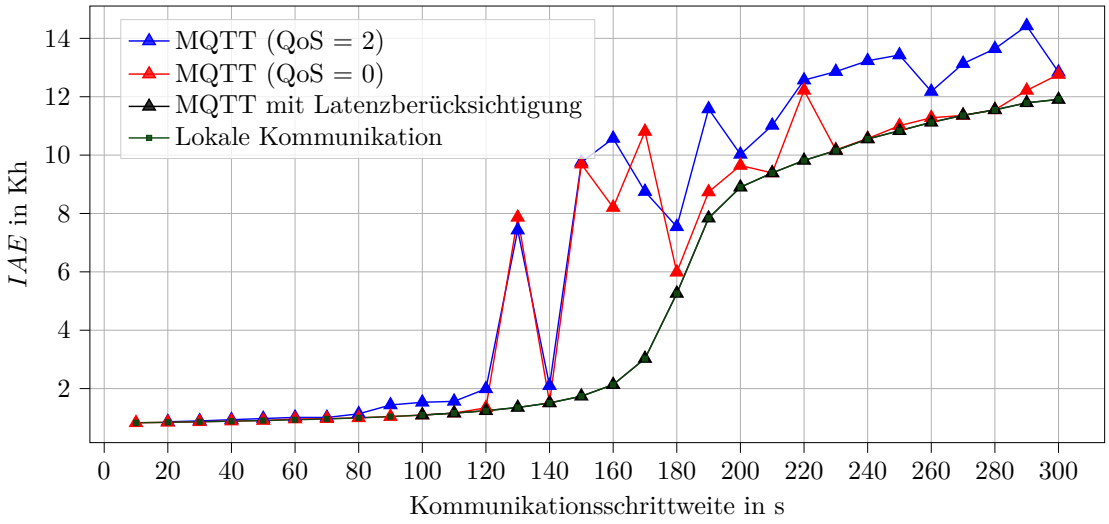


Abbildung 6.12: Ergebnisse der IAE für verschiedene Kommunikationsschrittgrößen und durchgeführte Studien

6.5 Entwicklung von Agenten

In diesem Kapitel werden einige weitere Module der *AgentLib* vorgestellt. Außerdem wird ihre Anwendung innerhalb der Agenten an Simulationsbeispielen gezeigt. Somit spiegelt dieses Kapitel Inhalte aus AP04 - Agenten und AP05 - Simulationen und Experimente wider.

6.5.1 Agentlib_MPC

MPC-Modul Ergänzend zu den strukturellen Änderungen in der *AgentLib* wurden umfassende Module zur Erstellung nichtlinearer modellprädiktiver Regler (MPC) erstellt. Diese Module wurden zu einem Plugin für die *AgentLib* zusammengefasst - *AgentLib_MPC*. Die Struktur des *MPC-Moduls* ist in Abbildung 6.13 zu sehen.

Das *MPC*-Modul (`agentlib_mpc.modules.mpc`) selbst implementiert die abstrakten Methoden des Basis-Moduls. Außerdem besitzt es Informationen über die Variablen des geregelten Systems und ihre Rolle, etwa ob es sich bei Eingängen des Regelstreckenmodells um Stör- oder Stellgrößen handelt, oder welchen Wert die Modellparameter annehmen. Es empfängt Messwerte, Prädiktionen für Störgrößen und Sollwerte, und sendet die berechneten Stellgrößen. Um in Zukunft verschiedene Optimierungsframeworks zu unterstützen, ist das *MPC*-Modul selbst ebenfalls modular aufgebaut und erlaubt so die kontinuierliche Erweiterung.

Die eigentliche numerische Optimierung der Stellgrößen geschieht im *OptimizationBackend* - einer funktionalen Klasse, die einem *MPC*-Modul bei der Instanziierung zugewiesen wird. Das *OptimizationBackend* besitzt zwei abstrakte Methoden:

- `setup_optimization()`
- `solve()`

Die Methode `setup_optimization()` erstellt das Optimierungsproblem basierend auf der Variablenstruktur und einem Modell, welche jeweils vom *MPC*-Modul übergeben werden. Zusätzlich können dem *OptimizationBackend* noch andere Konfigurationen übergeben werden, wie Zeitschrittweite, Prädiktionshorizont oder Optionen für den numerischen Lösungsalgorithmus. Die Methode `solve()` führt einen Optimierungsschritt der MPC aus. Als Eingang dient ein sog. *Dictionary*, welches eine Datenstruktur darstellt, die die Namen aller Modellvariablen mit ihren Werten verknüpft. Das *OptimizationBackend* muss diese Werte einlesen, gegebenenfalls auf das Diskretisierungsgitter interpolieren und geordnet dem Optimierer übergeben. Anschließend muss das Ergebnis der Optimierung wieder sortiert und den Stellgrößen zugeordnet werden.

Als numerische Basis für die Optimierung mit *Agentlib_MPC* wurde das Open Source Framework CasADi [Andersson et al., 2019] gewählt, mit dem beliebige nicht-lineare Differentialgleichungen formuliert und gelöst werden können. Außerdem können mit CasADi formulierte Modelle genutzt werden, um effizient nicht-lineare Optimierungsprobleme, die beispielsweise in der modellprädiktiven Regelung auftreten, zu lösen. Daher wurde als erstes *OptimizationBackend* ein *CasadiBackend* implementiert, welches sich auf CasADi stützt. Hierdurch können nun nicht-lineare Prozessmodelle optimiert werden, die als *CasadiModel* zur Verfügung stehen. Das entstehende Optimalsteuerungsproblem wird mit der direkten Kollokations-Methode diskretisiert. Das nicht-lineare Optimierungsproblem kann dann mit Algorithmen, wie z. B. IPOPT [Wächter and Biegler, 2006] oder qpOASES [Ferreau et al., 2014] (im Falle eines quadratischen Problems) gelöst werden.

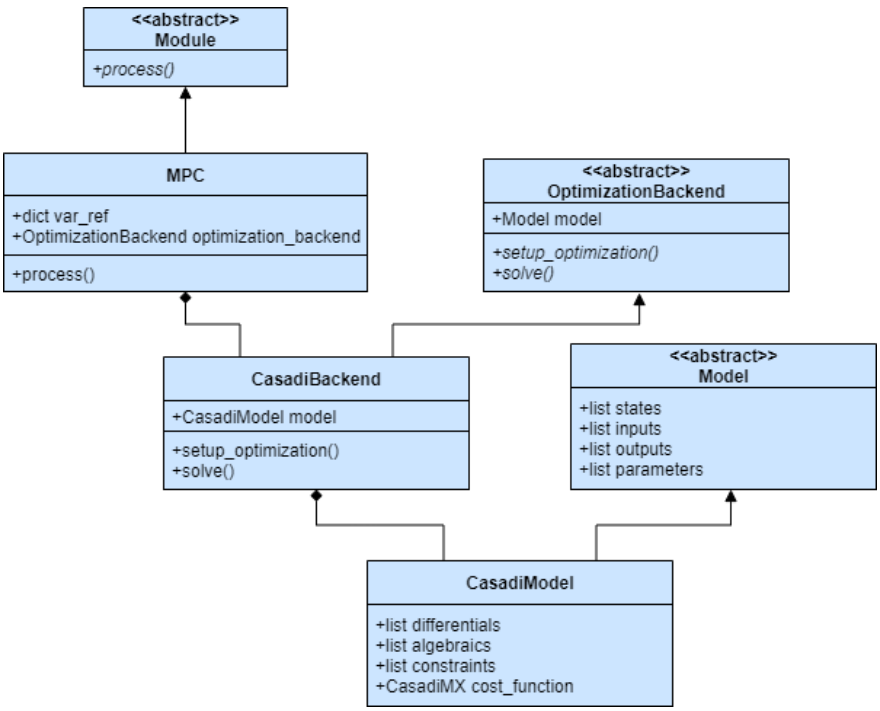


Abbildung 6.13: Klassenstruktur für das *MPC*-Modul in der *AgentLib*. Das MPC-Modul realisiert die Einbindung in das Agentensystem, während das *OptimizationBackend* die Lösung des Optimalsteuerungsproblems durchführt.

Um standardisiert differentiell-algebraische Systeme in CasADi-Syntax zu implementieren wurde die Modellklasse *CasadiModel* entwickelt. Die so implementierten Modelle sollen für zwei Zwecke nutzbar sein:

1. Simulation mit dem *Simulator*-Modul (*agentlib.modules.simulator*))
2. Anwendung im neu erstellten *MPC*-Modul

Um diese Anforderungen gleichermaßen zu erfüllen, besteht ein *CasadiModel* aus folgenden Bestandteilen:

- Deklaration aller Variablen und Parameter basierend auf dem FMI-Standard
- Formulierung der Differentialgleichungen
- Formulierung eventueller algebraischer Gleichungen (für Simulation und Optimierung)
- Formulierung von (Un-)Gleichheitsbedingungen (nur für die Optimierung)
- Formulierung einer Kostenfunktion (nur für die Optimierung)

In Auflistung 6.5 ist beispielhaft die Implementierung eines nichtlinearen Modells mit CasADi zu sehen. Das Modell repräsentiert eine simple Raumlüftung, bei der die Raumtemperatur über den kühlen Luftmassenstrom einer Lüftungsanlage geregelt wird. Das Modell enthält einen differenziellen Zustand, sowie mehrere Eingänge und Parameter (Listen hier gekürzt). Über eine weiche Beschränkung soll die Raumtemperatur unterhalb eines Maximalwertes gehalten werden.

Auflistung 6.5: Beispiel CasadiModel. Die Konfiguration wurde der Übersicht halber gekürzt.

```
class MyCasadiModelConfig(CasadiModelConfig):
    inputs: List[CasadiInput] = [
        CasadiInput(name="mDot_0", value=0.0225),
        CasadiInput(...),
        ...
    ]
    states: List[CasadiState] = [
        CasadiState(name="T_0", value=293.15),
        ...
    ]
    parameters: List[CasadiParameter] = [
        CasadiParameter(...),
        CasadiParameter(...),
    ]
    outputs: List[CasadiOutput] = [
        CasadiOutput(name="T_0_out")
    ]

class MyCasadiModel(CasadiModel):
    config_type = MyCasadiModelConfig
    def setup_system(self):
        self.T_0.ode = (
            self.cp * self.mDot_0 / self.c_0 * (self.T_in - self.T_0)
```

```
            + self.d_0 / self.c_0
        )
        self.T_0_out.alg = self.T_0
        self.constraints = [(0, self.T_0 + self.T_0_slack, self.T_0_upper)]
        objective = sum([
                self.r_mDot_0 * self.mDot_0,
                self.s_T_0 * self.T_0_slack**2,
            ])
        return objective
```

Ähnlich wie Module, besteht das Modell aus einer Konfigurationsklasse, die von `CasadiModelConfig` erbt, und einer Modellklasse, die von `CasadiModel` erbt. In der Konfigurationsklasse werden analog zum FMI-Standard Eingänge (`inputs`), innere Zustände (`states`), Parameter (`parameters`) und Ausgänge (`outputs`) definiert. In der Modellklasse werden nun die Differenzialgleichungen und Zuweisungen für die Zustände und Ausgänge definiert. Ebenfalls können Nebenbedinungen (`self.constraints`) und eine Zielfunktion definiert werden. Das Modell kann sowohl zum simulieren als auch für modellprädiktive Regler verwendet werden. Im Fall einer Simulation, werden Nebenbedingungen und Zielfunktion ignoriert.

In Auflistung 6.6 ist eine beispielhafte Konfiguration einer modellprädiktiven Regelung mit dem Modell aus Auflistung 6.5 zu sehen. Unter “type” type wird der Modultyp auf das MPC-Modul festgelegt, das aus dem Plugin `agentlib_mpc` stammt. Das “optimization_backend” wird zu `casadi` festgelegt, und der Pfad zur Modellimplementierung festgelegt. Schließlich können noch Prädiktionshorizont ("prediction horizon"), Schrittweite (“time_step”) und die relevanten Variablen festgeglegt werden. Hierbei sind “parameters” Werte, die fest über dem Prädiktionshorizont sind, “inputs” sind Eingäne, die variabel über den Prädiktionshorizont sind, “states” sind die Zustände, die über den Horizont integriert und optimiert werden, und “controls” sind die Stellwerte, die die MPC errechnen soll.

Auflistung 6.6: Konfiguration eines MPC-Moduls

```
agent_mpc = {
    "id": "myMPCAgent",
    "modules": [
        {"module_id": "Ag1Com", "type": "local_broadcast"},
        {
            "module_id": "myMPC",
            "type": "agentlib_mpc.mpc",
            "optimization_backend": {
                "type": "casadi",
                "model": {"type": {"file": __file__, "class_name": "↩
                    MyCasadiModel"}},
                "results_file": "results.csv",
            },
            "time_step": 900,
            "prediction_horizon": 5,
```

```
            "parameters": [
                {"name": "q_T_0", "value": 0},
                {"name": "s_T_0", "value": 3},
                {"name": "r_mDot_0", "value": 1},
            ],
            "inputs": [
                {"name": "d_0", "value": 150, "alias": "load_prediction"},
                {"name": "T_0_set", "value": 294.55},
                {"name": "T_0_upper", "value": ub, "interpolation_method": "↩
                    previous"},
                {"name": "T_in", "value": 290.15},
            ],
            "controls": [{"name": "mDot_0", "value": 0.02, "ub": 1, "lb": 0}],
            "states": [{"name": "T_0", "value": 298.16, "ub": 303.15, "lb": ↩
                288.15}],
        },
    ],
}
```

Mit der in Auflistung 6.6 beschriebenen MPC und dem in Auflistung 6.5 beschriebenen Modell, wurde zu Testzwecken folgendes Agentensystem ausgeführt:

- Prädiktionsagent mit folgenden Modulen
  - Prädiktionsmodul, verschickt prädizierte Lasttrajektorien
  - Sensormodul, verschickt leicht verrauschte Werte für die aktuelle thermische Last
  - Kommunikator
- Simulationsagent mit folgenden Modulen
  - Simulatormodul mit Modell aus Auflistung 6.5
  - Kommunikator
- MPC-Agent mit folgenden Modulen
  - MPC-Modul mit Konfiguration aus Auflistung 6.6
  - Kommunikator

Die Ergebnisse des Agentensystems sind in Abbildung 6.14 zu sehen. Auf der oberen Grafik ist die Raumtemperatur in schwarz, und die erlaubte Obergrenze für die Raumtemperatur in rot zu sehen. Die mittlere Grafik zeigt den Lüftungsmassenstrom, und die untere die thermischen Gewinne des Raums. Man erkennt, dass die MPC den Raum aufheizen lässt während der Zeiten mit höherer Temperaturgrenze, und zügig wieder den Luftmassenstrom erhöht, damit rechtzeitig abgekühlt wird für die schärfere Begrenzung.

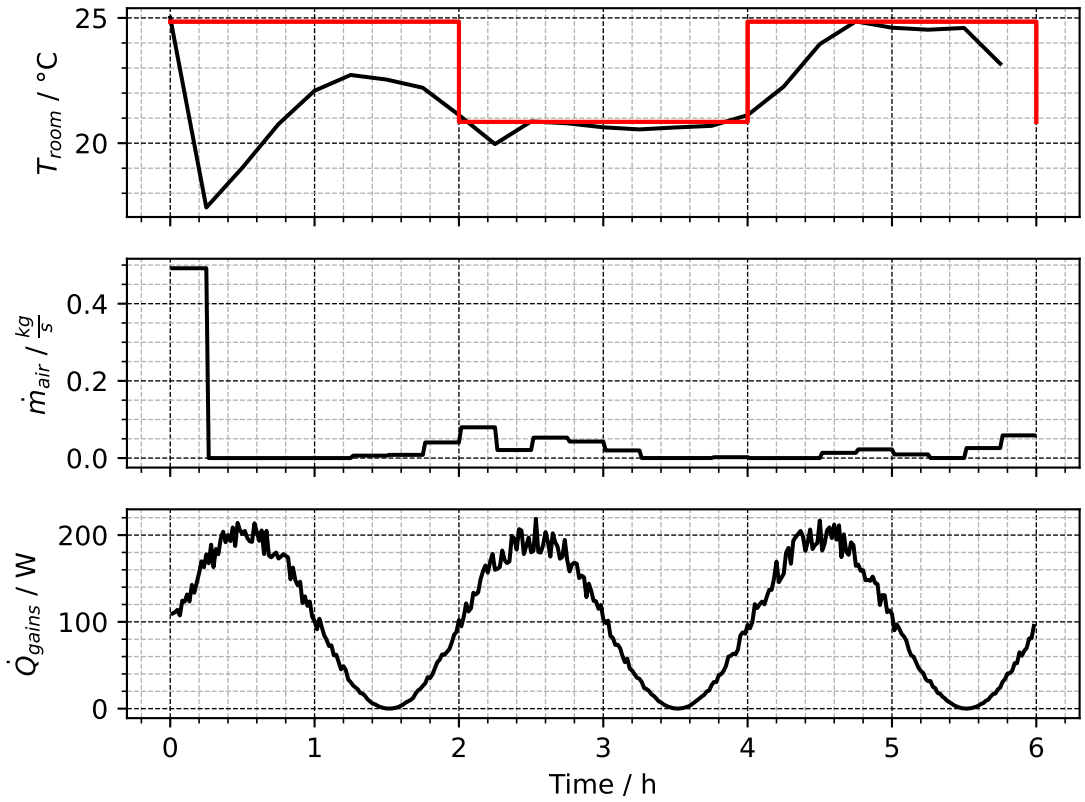


Abbildung 6.14: Beispiel modellprädiktive Regelung

6.5.2 Komfort-Agenten

AdaptiveComfort-Module Zu Beginn des AP6 wurde ein adaptives Komfortmodul basierend auf dem ASHRAE 884-RP Standard entwickelt. Der Standard definiert ein adaptives Modell für den thermischen Komfort, basierend auf der Außentemperatur der vorangegangen Woche. In der *AgentLib* wird das Paket *pythermalcomfort* [Tartarini and Schiavon, 2020] verwendet, um die kritischen Temperaturen zu bestimmen, bei denen 90 % Nutzerzufriedenheit gewährleistet ist. Das *AdaptiveComfort*-Modul der *AgentLib* kann diese Werte kontinuierlich berechnen und an Regelungsagenten senden, die die Raumtemperatur innerhalb dieses Bandes halten. In der *AgentLib* wurde ein entsprechendes Beispielsystem entwickelt, das ein Agentensystem mit Benutzung des entwickelten Komfortmoduls darstellt. Hierbei soll ein Raum mit einem Luftmassenstrom von konstanter Temperatur gekühlt werden, wobei ein Komfortagent die zulässigen Raumtemperaturen bestimmt. Abbildung 6.15 stellt den Aufbau des Agentensystems dar.
Der Wetteragent versendet regelmäßig die aktuelle Außentemperatur. Der Komfortagent nutzt diese, um adaptiv das zulässige Temperaturband zu bestimmen. Auf der anderen Seite prädiziert ein Lastagent den Wärmeeintrag in das System. Zusätzlich sendet er die tatsächliche Last an den

Simulationsagenten. Die tatsächliche Last ist dabei gegenüber der Prädiktion mit einem schwachen Rauschen versehen. In diesem einfachen Beispiel ist der Wärmeeintrag noch nicht von der Außentemperatur abhängig, für zukünftige Untersuchungen wäre dies jedoch sinnvoll. Mit Hilfe der Lastvorhersage, der Sollwerte und der aktuellen Raumtemperatur berechnet der MPC-Agent die optimale Steuerungsgröße und sendet diese an den Simulationsagenten, um den Regelkreis zu schließen.

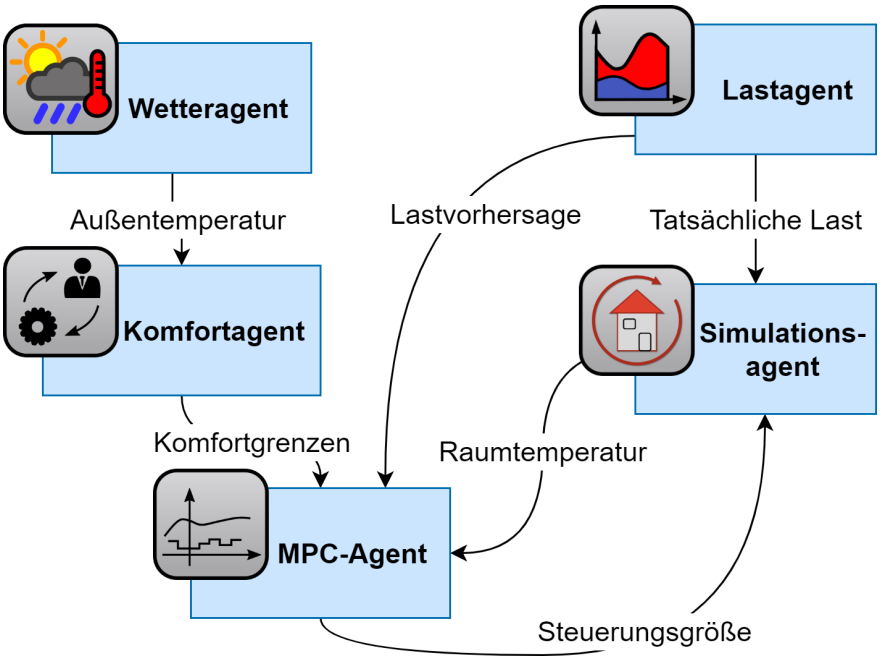


Abbildung 6.15: Struktur eines Agentensystems mit einer modellprädiktiven Regelung, die über das Komfort-Modul mit Soll-Werten versorgt wird.

In Abbildung 6.16 sind Simulationsergebnisse des gezeigten Agentensystems über einen zwölf Tage Zeitraum zu sehen. Jeden Tag bestimmt der Komfortagent neue Werte für die Grenzen der Raumtemperatur. Der modellprädiktive Regler hält die zulässigen Temperaturgrenzen größtenteils ein. Kleinere Verletzungen der Temperaturgrenzen ergeben sich aus der nicht perfekten Prädiktion des Wärmeeintrags.

Die Gültigkeit des ASHRAE 884-RP Standard ist jedoch nicht für alle Gebäude zulässig, insbesondere bei kälteren klimatischen Bedingungen. Außerdem konnten noch keine Untersuchungen zu Agenten, die adaptiv auf Nutzeranforderungen reagieren, getätigt werden. Aus diesen Gründen wurden weitere Komfortmodule in die *AgentLib* integriert.

PMV-Module Als ein weiteres Komfortmodul wurde ein Prozess zur Komfortbestimmung basierend auf dem Predicted-Mean-Vote (PMV) Ansatz entwickelt. Der PMV ist eine Kenngröße zwischen -3 und 3, die das durchschnittliche Behaglichkeitsbefinden unter gegebenen Bedingungen prädiziert. Dabei zeigen negative Werte einen zu kalten Raum an, während positive Werte einen zu warmen Raum anzeigen. Mithilfe des Python-Pakets *pythermalcomfort* [Tartarini and Schiavon,

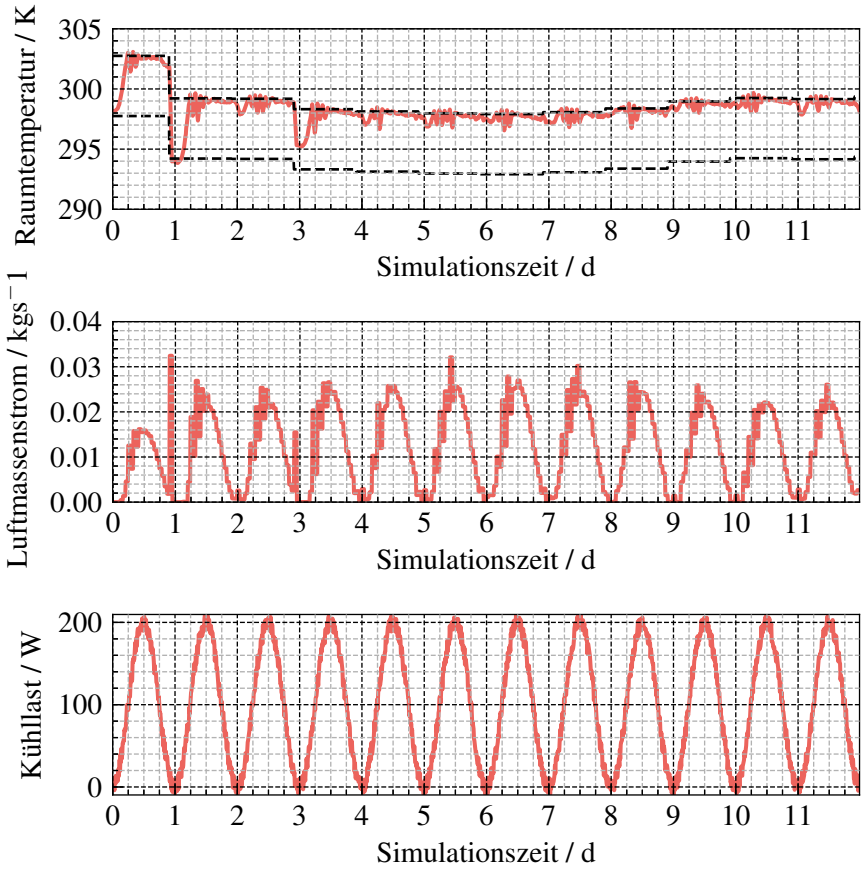


Abbildung 6.16: Simulationsergebnisse aus dem Agentensystem mit Komfortagent. Die gestrichelten schwarzen Linien zeigen das zulässige Komfortband.

2020] kann der PMV basierend auf der aktuellen Außentemperatur, der Raumtemperatur, der Luftgeschwindigkeit, der Feuchtigkeit und Koeffizienten für den Akivitätsgrad und die Bekleidung der sich in dem Raum befindenden Personen bestimmt werden.

Im Zentrum der neuen Komfortbestimmung steht ein neuer Komfortagent, der den PMV beziehungsweise ein nutzerspezifisches Äquivalent dazu bestimmen kann. Liegen dem Komfortagenten Feedback-Daten eines Nutzers vor, so kann er mithilfe des PMV-Rechners rekonstruieren, welche Abweichung das Komfortempfinden des Nutzers gegenüber dem PMV hat. Dies geschieht, indem der Mittelwert der Differenzen des Nutzerfeedbacks zu den dazugehörigen PMV-Werten gebildet wird und auf zukünftige Berechnungen zur Komfortbestimmung aufaddiert wird. In der Zukunft könnte man auch fortschrittlichere Methoden des maschinellen Lernens angewendet werden.

Aus dem Wissen über den Komfort einer bestimmten Raumumgebung muss nun noch eine konkrete Handlungsempfehlung für den Sollwert der Raumtemperatur abgeleitet werden. In der jetzigen Version des Komfortagenten geschieht dies mithilfe von Bisektion. So findet der Agent iterativ eine Sollwerttemperatur, bei der die Komfortbewertung zwischen -0,1 und 0,1 liegt.

Zur Demonstration der neuen Methodik wurde eine Simulation unter Einbezug des neuen Komfortagenten und eines Nutzeragenten durchgeführt. Die Struktur dieser Simulation ist in Abbildung 6.17 zu sehen. Im Gegensatz zum vorigem Beispiel wurde die Struktur um einen Nutzeragenten erweitert und der Komfortagent wurde ausgetauscht.

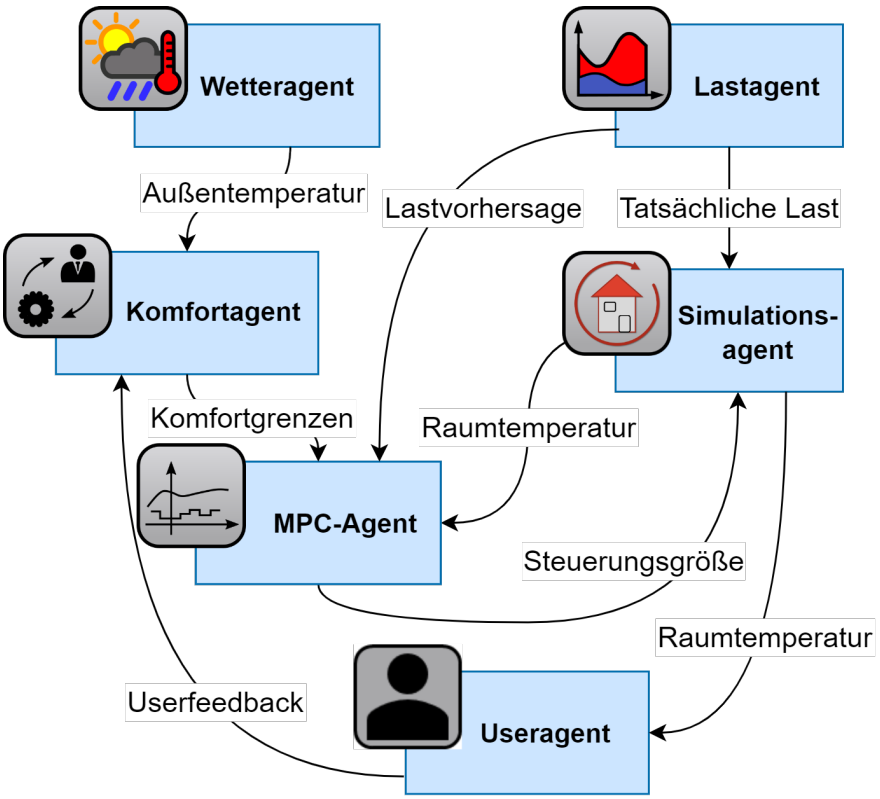


Abbildung 6.17: Struktur eines Agentensystems mit einer modellprädiktiven Regelung, die über das Komfort-Modul mit Soll-Werten versorgt wird. Das Komfortmodul lernt dabei aus Userfeedack.

Der Nutzeragent liefert sporadisch ein Feedback, wie der Nutzer das Empfinden im Raum einschätzt (-3 bis 3). Dazu wird im Nutzeragent zuerst der PMV berechnet und dann ein zufälliger Störwert aufaddiert, der jedoch für einen Nutzer eine Tendenz in eine Richtung hat. So lässt sich z.B. ein Nutzer emulieren, der es immer etwas kühler möchte als der Durchschnittsnutzer.

Der Komfortagent erhält dieses Feedback und speichert es. Solange nicht ausreichend Trainingsdaten vorliegen, berechnet der Komfortagent den PMV und liefert somit eine allgemeine Komfortbewertung. Wenn eine ausreichende Menge Daten verfügbar sind, wird die gemittelte Abweichung des Nutzerfeedbacks vom PMV berechnet und das Verhalten des Agenten passt sich an den Nutzer an. So passt es sich an die individuellen Bedürfnisse des Nutzers an und antizipiert, welches Unbehagen ein gegebenes Raumklima bei diesem speziellen Nutzer hervorruft. Dadurch kann die Sollwerttemperatur im Laufe der Simulation angepasst werden, um den Bedürfnissen des Nutzers zu entsprechen. Die Ergebnisse dieser Simulation sind in Abbildung 6.18 zu sehen.

Nach ca. 2 Tagen im Betrieb führt der Komfortagent die Berechnung der Abweichung durch und

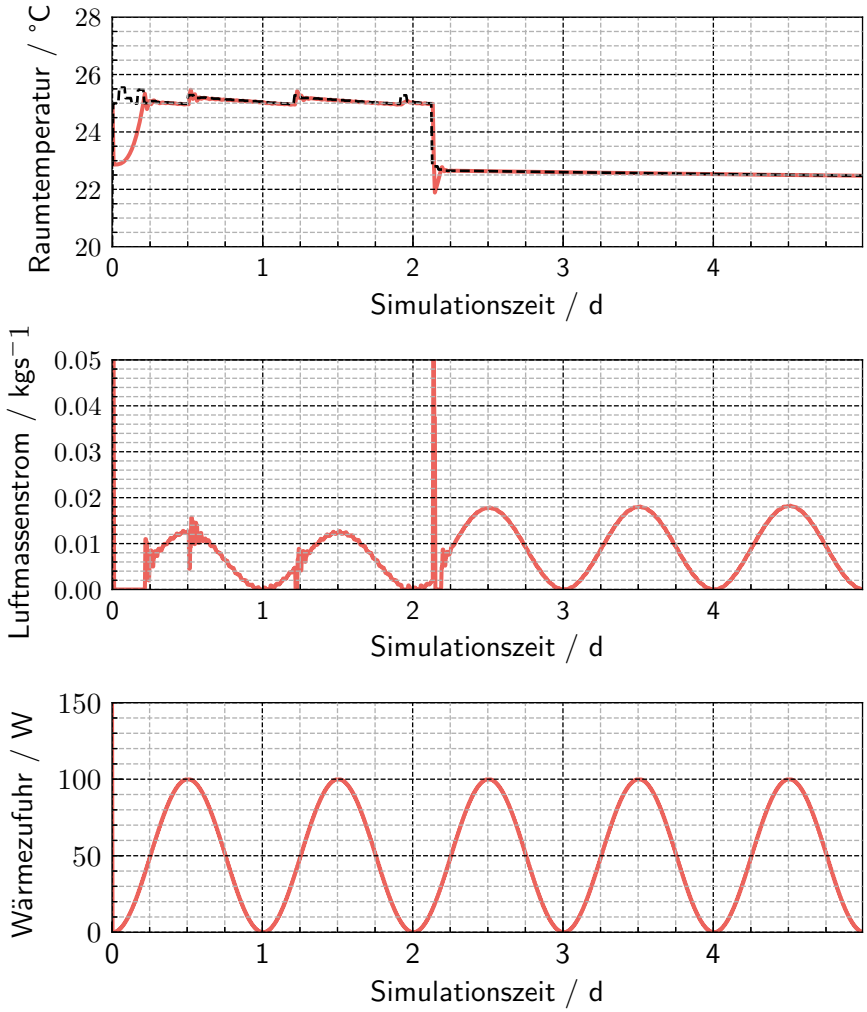


Abbildung 6.18: Simulationsergebnisse aus dem Agentensystem mit Komfortagent. Die gestrichelten schwarzen Linien zeigen den Sollwert, den der Komfortagent bestimmt.

erfüllt somit den Wunsch des Nutzers, den Raum kälter zu regeln. Ebenfalls zu erkennen ist die kontinuierliche Veränderung des Sollwerts, selbst wenn die Abweichung nicht neu berechnet wurde. Dies sind Nebeneffekte der Bisketionsmethode, die den Sollwert iterativ aus der Komfortbewertung bestimmt. Diese Effekte sind selbstverständlich nicht erwünscht und deuten darauf hin, dass die Methodik zukünftig noch verfeinert werden sollte.

Implementierung des thermophysiologischen Komfortmodells „NOODEL“ In Hinblick auf die immer weiter an Relevanz gewinnenden dynamischen Regelstrategien von Nichtwohngebäuden mithilfe von beispielsweise modellprädiktiven Regelungen müssen insbesondere auch transiente thermische Randbedingungen bei der Bewertung der thermischen Behaglichkeit betrachtet werden. Herkömmliche Modelle, wie das zuvor vorgestellte AdaptiveComfort-Module und der PMV-Index basieren auf der Annahme, dass sich die Gebäudenutzenden zu jeden Zeitpunkt im thermischen

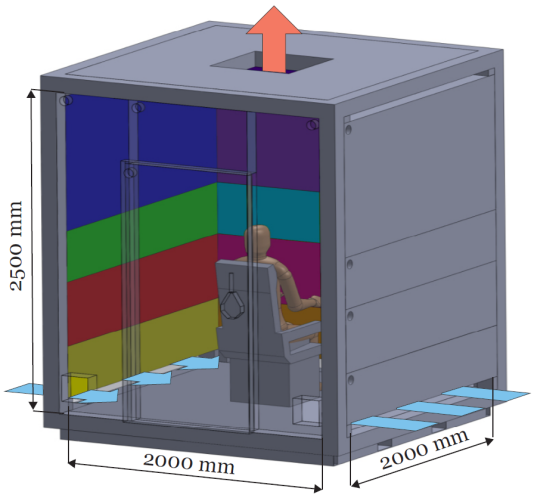
Gleichgewicht mit der Umgebung befinden. Insbesondere unter transienten Randbedingungen, wie z.B. bei Temperaturanstiegen oder -abfällen, führt diese Annahme beim AdaptiveComfort-Module und dem PMV-Index zu deutlichen Ungenauigkeiten bei der Bewertung des thermischen Komforts. Um auch unter transienten Randbedingungen präzise Komfortvorhersagen zu erhalten, wurde die Einbindung eines thermophysiolgischen Komfortmodells vorangetrieben.
Thermophysiolgischen Komfortmodelle bestehen in der Regel aus einem passiven und einem aktiven System des menschlichen Körpers. Durch das passive System wird der Wärmetransport zwischen Mensch und Umgebung und innerhalb des Mensch-Modells abgebildet. Die Modelle des passiven Systems lassen sich zunächst hinsichtlich der Abbildung des Körpers in ein oder mehrere Körper-Segmente aufteilen. Die weitere Gliederung erfolgt nach der Anzahl an Schichten bzw. Knoten mit der jedes Segment aufgelöst wird. Das aktive System beschreibt die Wärmeregulation innerhalb des menschlichen Körpers. Dieses wird durch ein virtuelles zentrales Nervensystem gesteuert und setzt sich aus den Funktionen zur Regulierung des Blutflusses über Gefäßverengung und Gefäßweitung sowie aus Schwitzen und Kältezittern zusammen. Ziel ist dabei die Aufrechterhaltung einer konstanten Körperkerntemperatur. Die Modelleingangsgrößen sind die Randbedingungen der Umgebung, sowie physiologische Parameter wie Bekleidung, metabolische Rate, Körpergewicht- und Größe, Fettanteil, Alter und Geschlecht. Als Ergebnis werden für die jeweiligen Segmente die Kern- und Hauttemperatur berechnet. Durch die Anbindung an ein psychologisches Modell können die berechneten Temperaturwerte zur Bewertung der thermischen Umgebung hinsichtlich einer subjektiven Skala interpretiert werden. Das Gesamtmodell kann als thermophysiologisches Komfortmodell bezeichnet werden.
Am Lehrstuhl für Gebäude- und Raumklimatechnik wurde von Streblow [2010] basierend auf den Arbeiten von Tanabe et al. und Zhang et al. das „33 Knoten Komfortmodell“ als 33NCM (33 Node Comfort Model) im Rahmen ihrer Dissertation entwickelt. Bei dem Modell handelt es sich um ein Multisegment-Modell, bei dem der Körper in 16 Segmente aufgeteilt ist. Die Segmente wiederum werden jeweils über 2 Knoten, eine Kern- und eine Hautschicht, modelliert. Alle Segmente sind über einen zentralen Blutknoten miteinander verbunden. Seit der ersten Entwicklungsversion wurde stetig an dem Modell weitergearbeitet. Die letzte Überarbeitung inklusive Kalibrierung und Validierung erfolgte im Jahr 2018. Diese Version von Streblow and Müller [2018] wurde unter dem Namen „NOODEL“ als Open Source Projekt auf „github“ publiziert, siehe auch https://ebc-tools.eonerc.rwth-aachen.de/noodel. Weiterhin wurden dem Modell im Jahr 2020 im Rahmen einer Dissertation von Rewitz [2020] zusätzliche Individualisierungsoptionen hinsichtlich Geschlecht, Alter und Köpermassenzusammensetzung hinzugefügt sowie die Möglichkeit, die Kalibrierung des Modells mit weiteren experimentellen Probandendaten modular zu erweitern. Das NOODEL-Modell eignet sich daher um auch die individuellen Eigenschaften der Nutzenden abbilden zu können und somit eine hohe Prädiktionsgenauigkeit des thermsichen Empfindens der Nutzer zu erzielen.
Für die Bewertung des thermischen Komforts benötigt das Komfortmodell diverse Eingangsgrößen. Dazu gehören Eingangsgrößen, die körperteilindividuell vorgegeben werden können: Luft- und

Strahlungstemperatur, Luftgeschwindigkeit, Kontaktflächentemperatur und solare Einstrahlung. Des Weiteren können körperteilindividuelle Parameter eingestellt werden, wie der Kontaktflächenanteil der Haut zu Oberflächen, der Kontaktflächenanteil der Haut zur solaren Einstrahlung und Isolationswerte für Bekleidungsstücke. Zusätzlich wird noch die körperliche Arbeit der Person, die globale relative Feuchte im Raum und die Information, ob die Person sitzt oder steht, vorgegeben. Als individuelle Parameter können die Körpergröße- und gewicht, Geschlecht sowie Alter vorgegeben werden.
Basierend auf den Eingangsgrößen berechnet das NOODEL-Modell den aktuellen thermischen Zustand des Körpers und darauf basierend sowohl die körperteilaufgeslösten Bewertungen des thermischen Empfinden und des thermischen Komforts. Das NOODEL-Modell ist in der Modellierungssprache Modelica implementiert, sodass eine Einbindung des NOODEL-Modells in die Agenten-Umgebung über den FMI-Standard erfolgt. Der Export des NOODEL-Modells als Functional-Mockup-Unit erfolgt als vollparametrisierbares Modell, sodass auch die nachträgliche Anpassung von individuellen Parametern, wie Geschlecht, Alter und Köpermassenzusammensetzung erfolgen kann. Das NOODEL-Modell kann somit als FMU-Simulator-Instanz in die Agenten-Umgebung eingebunden werden.

6.5.3 Integration der Komfortprüfkammer „ACCu“

Für zukünftige Tests und Validerungen von Komfortagenten und Nutzermodellen wurde außerdem eine Komfortprüfkammer, der Aachen Komfort Kubus (ACCu, siehe Abbildung 6.19), in die AgentLib eingebunden. Der ACCu wurde am EBC von Möhlenkamp [2019] entwickelt und ist ein hochmodularer Komfortprüfstand, in dem unterschiedliche Innenraumsituationen abgebildet werden können. Der Komfortkubus hat eine Grundfläche von 2 m x 2 m und eine Höhe von 2,5 m. Die Umschließungsflächen sind horizontal segmentiert, so dass sich insgesamt 16 Flächensegmente ergeben, die individuell geheizt und gekühlt werden können. Durch die individuell steuerbaren Flächensegmente können thermische Asymmetrien und vertikale Temperaturgradienten von bis zu 10 K/m präzise eingestellt werden. Mit einem variablen Lüftungssystem können Luftwechselraten von bis zu 60 $\frac{\mathrm{m}^3}{\mathrm{h}}$ mit einer regelbaren Zulufttemperatur und –feuchte realisiert werden. Die Zuluft kann sowohl in Form einer Quelllüftung als auch in Form einer Mischlüftung in den Komfortkubus eingebracht werden. So können unterschiedlichste Umgebungsbedingungen und Innenraumsituationen abgebildet und sowohl objektiv als auch subjektiv durch Probanden hinsichtlich des thermischen Komforts und der thermischen Behaglichkeit bewertet werden. Die Steuerung des ACCu wurde in der Software LabView der Firma National Instruments programmiert und wurde im Rahmen des Projektes mit einer MQTT-Schnittstelle ausgestattet, die es erlaubt, Ist- und Sollwerte mit beliebigen Agenten-Modulen über einen MQTT-Broker auszutauschen. Die MQTT-Schnittstelle des ACCu wurde so konzipiert, dass eine direkte Kompatibilität mit dem JSON-Format einer Agenten-Variable gegeben ist. Somit können Ist-Werte aus dem Innenraum des ACCu, wie z.B. Luft- oder Strahlungstemperatur, direkt als Eingangsgrößen in Agenten-Modulen, wie z.B. Komfort- oder Nutzermodellen, genutzt werden. Aus den Ausgangsdaten der Komfort- oder Nutzermodelle kön-

Außenansicht (Foto: Winandy)



Flächensegmentierung und Luftführung bei Quelllüftung

Abbildung 6.19: Aufbau der Komfortprüfkammer ACCu

nen wiederum neue Sollwerte, wie z.B. Zulufttemperaturen abgeleitet werden, um entsprechende Umgebungsbedingungen im ACCu einregeln zu können. In der Steuerung des ACCu wurde ein „Agenten-Modus“ integriert, der bei Aktivierung alle für Komfort-Agenten relevante Istwerte in einer individuell einstellbaren Frequenz an einen MQTT-Broker sendet und gleichzeitig Sollwerte aus abonnierten Topics aus dem Communicator-Modul des Agentensystem übernimmt. Die Auswahl der Sollwerte erfolgt über die Abonnierung von entsprechenden Topics in der ACCu-Steuerung. Die Topics sind als sogenannter Topic-Tree mit mehreren Levels aufgebaut. Auf Seiten der Agenten-Regelung werden Sollwerte mit einem 2-Level Topic an den MQTT-Broker gesendet. Auf dem oberen Level wird ein entsprechender Name gewählt, aus dem eindeutig hervorgeht, dass diese Topics nur für den „ACCu-Agenten“ relevant sind. Im zweiten Level werden die einzelnen Sollwerte mit entsprechenden Aliassen benannt. Ein Topic würde dann z.B. „ACCuAgent/T_Zuluft_Soll“ heißen. Im oberen Level „ACCuAgent“ wird definiert, dass diese Sollwerte für den ACCu-Agenten relevant sind. „T_Zuluft_Soll“ definiert das Alias für den entsprechenden Sollwert. In der MQTT-Schnittstelle der ACCu-Steuerung wurde eine eindeutige Zuordnung der Sollwerte-Aliasse und der internen Variablen-Bezeichnungen integriert. So können beispielsweise alle Messages, die auf dem obere Topic-Level „ACCu-Agent“ versendete werden, abonniert werden. Eine empfangene Nachricht kann dann durch das Variablen-Alias in der MQTT-Schnittstelle der ACCu-Steuerung eindeutig zugeordnet werden und der Sollwert eingestellt werden. Hierbei ist zu beachten, dass auf Seiten der Agenten-Regelung die Aliasse korrekt gewählt werden müssen. Falsch gewählte bzw. der ACCu-Steuerung unbekannte Variablen-Aliasse werden ignoriert. Alternativ können auch nur einzelne Variablen abonniert werden, falls dies erforderlich sein sollte. Der ACCu kann somit zukünftig genutzt werden, um entwickelte Komfortbasierte Agenten-Systeme experimentell zu testen und zu validieren.

Demonstration einer komfortbasierten Zuluftregelung im ACCu auf Basis des NOODEL-Komfortmodells

Die Implementierung des NOODEL-Komfortmodells in die AgentLib und die MQTT-Schnittstelle des ACCu wird im Folgenden in einem Use-Case demonstriert. Die Stuktur des Use-Case wird in Abbildung 6.20 dargestellt. Im Zentrum des Agentensystems steht dabei der ACCu. Ziel dieses Versuches ist es, für zwei virtuelle Insassen im ACCu thermische Randbedingungen so herzustellen, dass ein Zielwert für das simulierte thermische Empfinden eingeregelt wird. Das thermische Empfinden der Insassen wird durch zwei Instanzen des NOODEL-Modells simuliert, welche zusammengefasst in Abbildung 6.20 als „NOODEL-Agent“ bezeichnet werden. Als Eingangsbedingungen des NOODEL-Modells werden die gemessenen thermischen Randbedinungen (Luft-/Strahlungstemperatur, relative Feuchte etc.) im ACCu genutzt. Das aus der Komfort-Simulation errechnete allgemeine thermische Empfinden der beiden NOODEL-Instanzen wird arithmetisch gemittelt und als Eingangsgröße dem „PID-Agenten“ übergeben. Der „PID-Agent“ berechnet einen entsprechenden Sollwert für die Zulufttemperatur, welcher wiederrum als Sollwert vom „ACCu-Agenten“ an die Steuerung des ACCu vorgegeben wird. Als Störgröße wird eine zeitliche Variation der Wandtemperaturen des ACCu vom „Last-Agent“ vorgegeben. Dadurch wird in diesem Anwendungsfall der Tagesgang der Außentemperaturen simuliert, was zu einer Änderung der Wandtemperaturen führt. Die thermischen Bedingungen im ACCu setzen sich dementsprechend aus den Wand- und den Zuluftkonditionen und der Wärmelast aus den beiden thermischen Manikins zusammen. Das Ziel ist es, dass Temperaturänderungen, hervorgerufen durch die variable Wandtemperatur, durch eine Anpassung der Zulufttemperatur kompensiert werden, um den Zielwert des gemittelten thermischen Empfindens zu erreichen.

Der experimentelle Aufbau im Innenraum des ACCu ist in Abbildung 6.21 dargestellt. Zur Vorbereitung der Messung wird der ACCu mit zwei Sitzplätzen ausgestattet und auf den Sitzplätzen wird je ein thermischer Manikin platziert, die jeweils einen konstanten Wärmestrom von etwa 100 W abgeben, um die Wärmeabgabe des Menschen an die Raumluft zu simulieren. Des Weiteren sind in Abbildung 6.21 die 4 Messpositionen der Lufttemperaturmessungen und der Strahlungstemperaturmessung abgebildet. An den Messpositionen A bis D ist jeweils ein vertikaler Strang an Pt100-Temperatursensoren auf den Höhen 0,1 m, 0,6 m, 1,1 m und 1,7 m angebracht. In Strang G sind mit der gleichen Höhenverteilung Globetemperatursensoren angebracht. Diese bestehen aus Pt100-Temperatursensoren, die jeweils von schwarz lackierten Kunststoffkugeln mit einem Durchmesser von 40 mm umgeben sind. Mithilfe der Lufttemperatur und der Globetemperatur kann die Strahlungstemperatur auf jeder Höhe anhand von Gleichung 6.1 nach Glück [2006] berechnet werden. In jeder Höhe wird die Strahlungstemperatur anhand der gemittelten jeweils aus den vier Strängen gemittelten Lufttemperatur und der Globetemperatur der jeweiligen Höhe berechnet.

$$T_{Strahlung,i} = 2.5 \cdot T_{Globe,i} - 1.5 \cdot T_{Luft,i} \tag{6.1}$$

Die Temperaturmessstellen werden den Eingangstemperaturen der einzelnen Körperteile der bei-

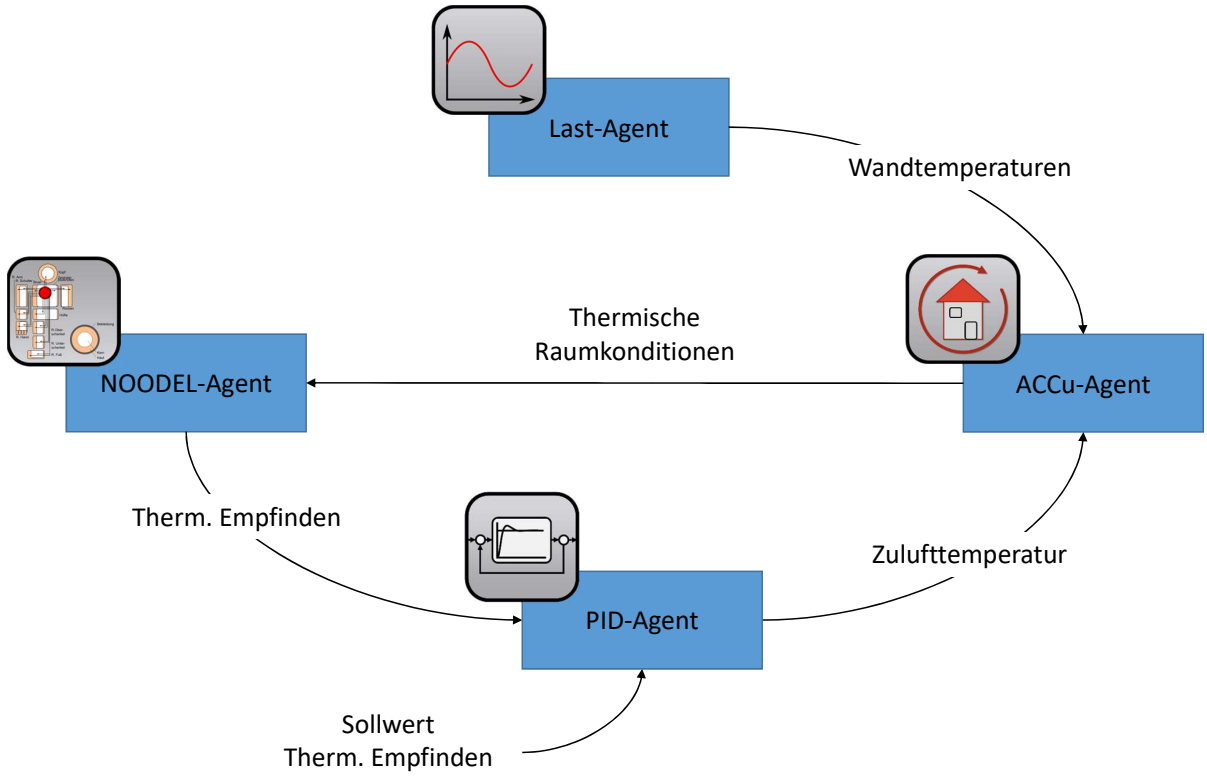
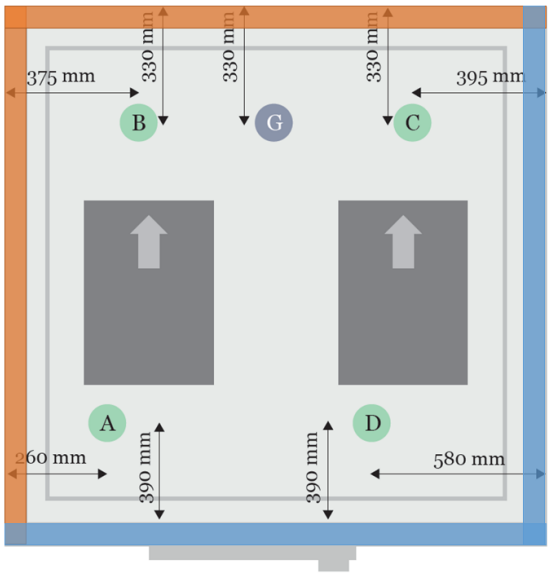


Abbildung 6.20: Struktur einer komfortbasierten Zuluftregelung für die ACCu-Komfortprüfkammer. Ein PID-Regler reglet die Zulufttemperatur basierend auf der simulierten thermischen Behaglichkeit auf einen Zielwert.

den NOODEL-Instanzen zugeordnet. In horizontaler Dimension werden die Lufttemperaturen der Stränge A und B für den linken Probanden und der Stränge C und D für den rechten Probanden gemittelt. In vertikaler Dimension werden die Luft- und Strahlungstemperaturen entsprechend der Tabelle auf der rechten Seit in Abbildung 6.21 den Körperteilen zugeordnet.
In einem Vorversuch wird die Luftgeschwindigkeit bei einem Volumenstrom von 200 $m^3\,h^{-1}$ gemessen, was zu einer maximalen Luftgeschwindigkeit im Innenraum von 0,1 $m\,s^{-1}$ führt. Da in diesem Luftgeschwindigkeitsbereich die Wärmeübertragung zwischen dem menschlichen Körper und der Raumluft durch die natürliche Konvektion deutlich höher ist als die durch die Raumluftströmung hervorgerufene erzwungene Konvektion, kann auf eine aktive Messung der Luftgeschwindigkeit während der Messkampagne verzichtet werden. Des Weiteren wird in einem Vorversuch ein manuelles Tuning des PID-Regler vorgenommen. Dabei werden die Wandsolltemperaturen auf 22 °C eingestellt und und der Zuluftvolumenstrom auf 200 $m^3\,h^{-1}$ eingestellt. Die Zulufttemperatur kann vom PID-Regler in einem Bereich von 16 °C bis 28 °C eingestellt werden.

In dem Demonstrationsexperiment werden die Wandtemperaturen auf 22 °C eingestellt. Die variablen Wände bekommen vom Lastagenten einen sinusförmigen Sollwertverlauf mit einem Mittelwert



Körperteil	Höhe in m
Füße	0,1
Unterschenkel	0,1
Oberschenkel	0,6
Becken	0,6
Rücken	0,6
Brust	0,6
Hände	0,6
Unterarme	0,6
Schulter	1,1
Oberarme	0,6
Kopf	1,1

Abbildung 6.21: Übersicht der Temperaturmesspositionen im ACCu. Die Stränge A - D bestehen aus jeweils vier Temperatursensoren und Strang G besteht aus 4 Globe-Temperatursensoren auf vier Höhen. Es sind außerdem die Wandsegmente markiert, die einen zeitlich variablen Oberflächentemperatursollwert (orange) oder einen zeitlich konstanten Oberflächentemperatursollwert (blau) vom Lastagenten vorgegeben bekommen. In der Tabelle sind die Zuordnungen der vertikalen Temperatursensorpositionen zu den einzelnen Körperteilen aufgelistet.

von 22 °C, einer Amplitude von 2 K und einer Periodendauer von 4 h eingestellt. Die Zuluft wird mit einem Volumenstrom von 200 $m^3\,h^{-1}$ in den Innenraum des ACCu über den Deckendralleinlass eingebracht. Dies entspricht einem für den Gebäudebereich recht hohen personenbezogenen Luftwechsel von 10 h^{-1}. Die personenbezogenen Parameter der beiden NOODEL-Instanzen sind in Tabelle 6.1 aufgelistet. Die Ergebnisse des Demonstrationsexperimentes sind in Abbildung 6.22

Parameter	Proband links	Proband rechts
Geschlecht	Männlich	Weiblich
Alter	30	30
Größe	1,80 m	1,70 m
Gewicht	75 kg	60 kg
Bekleidungswert Oberkörper	0,3 clo	0,3 clo
Bekleidungswert Unterkörper	0,3 clo	0,3 clo
Bekleidungswert Schuhe	0,05 clo	0,05 clo

Tabelle 6.1: Personenbezogene Parameter der beiden im ACCu simulierten NOODEL-Instanzen

dargestellt. Im oberen Diagramm werden die Temperaturverläufe der Wandoberflächen über den Versuchszeitraum von 5 h dargestellt. Nach einer Einschwingphase von 30 min beginnt die sinusförmige Variation der Oberflächentemperatur der variablen Wände von 22 °C auf 20 °C und auf 24 °C, während die konstanten Wände über die komplette Dauer des Experiments eine Temperatur

von 22 °C aufweisen. Im darunterliegenden Diagramm sind die Soll- und Istzulufttemperatur dargestellt. Nach der Einschwingphase von 30 min liegt eine Zulufttemperatur von ca. 23 °C vor. Nach der Absenkung der Wandtemperaturen erfolgt ein Anstieg bis auf fast 24 °C. Nach dem Anstieg der Wandtemperaturen nach ca. 1,5 h beginnt eine kontinuierliche Absenkung der Zulufttemperatur bis diese nach ca. 4 h wieder ansteigen. Im nachfolgenden Diagramm sind die Raumtemperaturen dargestellt. Dabei sind die Lufttemperaturen der Stränge A und B gemittelt und als „Luft links“ bezeichnet und die Lufttemperaturen der Stränge C und D ebenfalls gemittelt und als „Luft rechts“ bezeichnet. Die Strahlungstemperatur ist die gemittelte Strahlungstemperatur des Strang G. Die Temperaturen folgen prinzipiell dem Verlauf der Wandtemperaturen jedoch in stark abgedämpfter Form. Im untersten Diagramm sind die simulierten thermischen Empfinden der beiden NOODEL-Instanzen sowie der Mittel- und Sollwert aufgetragen. Das thermische Empfinden des NOODEL-Modells wird als eine stetige Bewertung von -3 bis $+3$ ausgegeben und steht für das allgemeine, globale thermische Empfinden des Probanden. Ein Wert von -3 steht hierbei für die Bewertung „kalt“, ein Wert von $+3$ für eine Bewertung von „warm“ und ein Wert von 0 für ein neutrales thermisches Empfinden, also „weder warm noch kalt“. Für das simulierte thermische Empfinden werden Werte von ca. 0 erreicht.
Im Laufe des Versuches führt die Variation der Wandtemperaturen zu einer leichten Absenkung bzw. zu einem leichten Anstieg der Luft- und Strahlungstemperatur im ACCu. Die daraus folgende geringfügige Änderung des simulierten thermischen Empfindens wird vom PID-Regler durch eine Änderung der Zulufttemperatur kompensiert. Prinzipiell stellt der daraus resultierende Verlauf der Zulufttemperatur einen leicht zeitlich versetzten, gespiegelten Verlauf der Wandtemperatur dar. Infolgedessen wird eine Luft- und Strahlungstemperatur von etwa 23 °C eingestellt, was unter den restlichen Randbedingungen (Bekleidungsgrad, personenbezogene Parameter) zu einer simulierten Bewertung des thermischen Empfindens von ca. 0 führt. Es ist außerdem erkennbar, dass geringfügige Unterschiede zwischen den beiden Probanden in der thermischen Behaglichkeit vorliegen. Dies ist sowohl aufgrund der geringfügigen Differenzen in der Lufttemperatur zwischen der linken und der rechten Seite im ACCu, sowie aufgrund der unterschiedlichen personenbezogenen Parameter zu erklären. Hierbei sei zu beachten, dass sich bei Änderung von Randbedingungen, wie z.B. Sonneneinstrahlung, erhöhte Luftgeschwindigkeit oder andere Bekleidungsparameter, auch die Luft- und Strahlungstemperatur ändert, die erforderlich ist, um ein angestrebtes thermisches Empfinden zu erreichen.

Im Rahmen dieser Untersuchungen konnte gezeigt werden, dass das NOODEL-Komfortmodell prinzipiell geeignet ist, um in einer modellbasierten Komfortregelung Anwendung zu finden. Neben der hohen Individualisierbarkeit können auch dynamische Änderungen der thermischen Randbedingungen gut hinsichtlich des thermischen Empfindens und der thermischen Behaglichkeit bewertet werden. In zukünftigen Arbeiten kann das NOODEL-Modell auch mit Gebäudesimulationen kombiniert werden, um gemeinsam in modellprädiktiven Regelungen genutzt werden zu können. Des Weiteren konnte die Komfortprüfkammer ACCu mit einer MQTT-Schnittstelle ausgestattet wer-

den, um eine Kommunikation mit Agenten-Modulen zum Austausch von Soll- und Istwerten zu realisieren. In zukünftigen Arbeiten können beispielsweise Gebäudesimulationen genutzt werden, um dynamische Änderungen in den Wandtemperaturen vorzugeben und entsprechende komfortbasierte Regelungen zu testen. Des Weiteren können Nutzer-Schnittstellen im ACCu in Form von digitalen Bewertungssystemen in die Agenten-Umgebung eingebaut werden, um zusätzliche Komfortbewertungen von Probanden zu verarbeiten und somit lernfähige Komfortmodelle, wie in Abschnitt 6.5.2 vorgestellt, mit weiteren Daten speisen können.

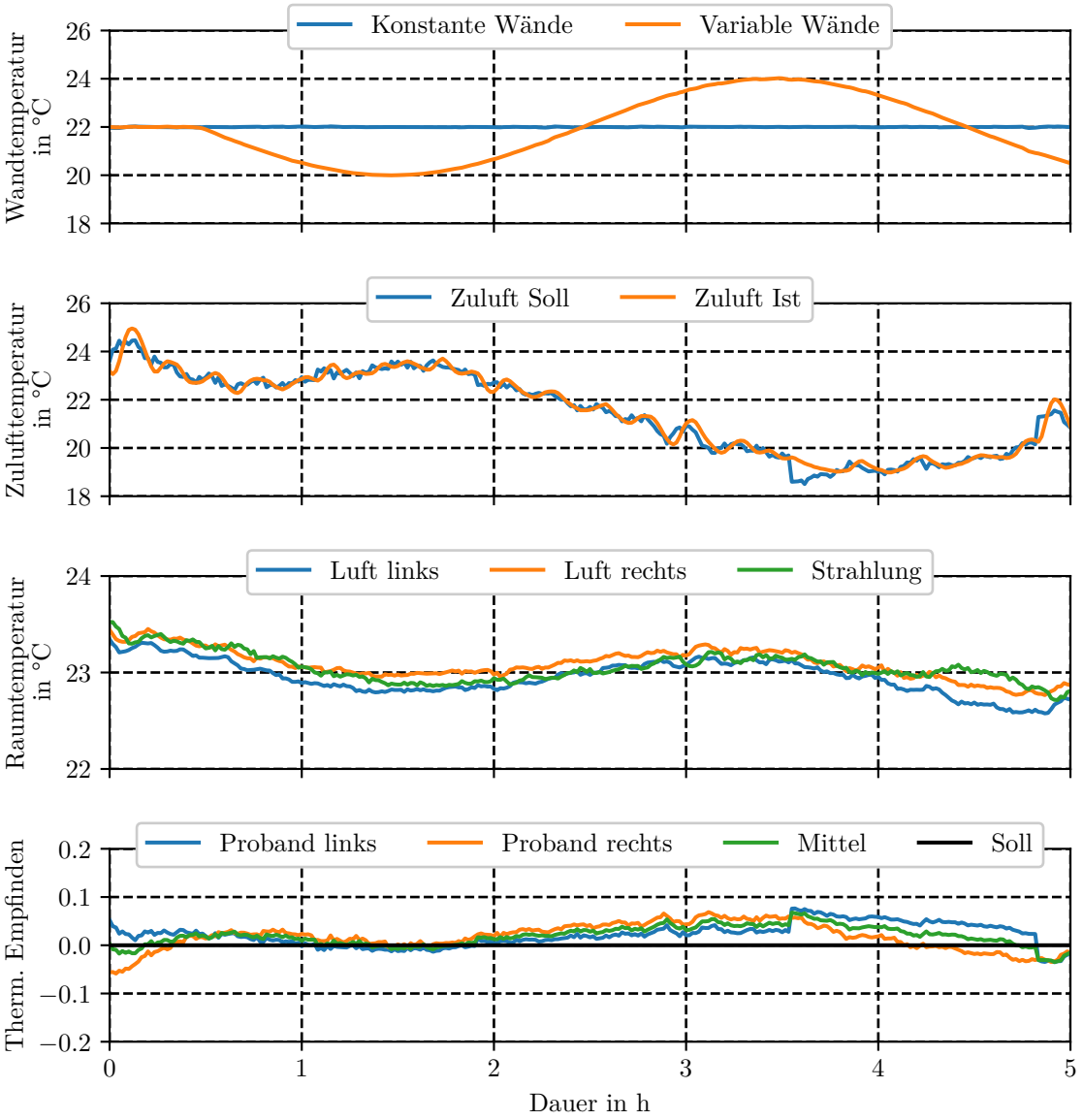


Abbildung 6.22: Ergebnisse des Demonstrationsexperimentes einer komfortbasierten Zulufttemperaturregelung im ACCu.

6.6 Verteilte modellprädiktive Regelung

Im Rahmen des Projektes wurden mehrere verteilte Regelungsansätze untersucht und implementiert und deren Eignung für Agentenbasierte Automationssysteme abgeleitet. Erstens wurde ein hierarchischer modellprädiktiver Regelungsansatz betrachtet. Hierdurch kann auf eine einfache Art eine begrenzte Resource auf mehrere Agenten verteilt werden. Zweitens wurde die Alternating Direction Method of Multipliers (ADMM) aufgegriffen und für sogenannte Konsensprobleme in die *AgentLib* implementiert. Drittens wurde eine sensitivitätsbasierte modellprädiktive Regelung implementiert. Dieser Ansatz erlaubt sowohl Kopplungen im Kostenfunktional als auch den Beschränkungen und kann daher vielseitig verwendet werden. Zuletzt wurde eine verhandlungsbasierte modellprädiktive Regelung entwickelt, bei der sich die Agenten auf die angewendeten Stellgrößen einigen. Für diese vier Ansätze werden im Folgenden die grundlegenden Ideen vorgestellt.

6.6.1 Hierarchische modellprädiktive Regelung

Bei einem hierarchischen DMPC Ansatz gibt es mehrere Ebenen. Beispielsweise wird auf der unteren Ebene die Strecke mittels mehreren lokalen modellprädiktiven Reglern auf Sollwerte geregelt. Die Sollwerte wiederum werden in der oberen Ebene mit einem vereinfachten Modell für mehrere bzw. alle unterlagerten Regler bestimmt.

Für die Regelung der Temperatur in Gebäuden kann für jede Zone jeweils ein modellprädiktiver Regler verwendet werden. Die Regler berechnen dabei unabhängig voneinander den gewünschten Frischluftmassenstrom. Die gewünschten Massenströme werden an einen übergeordneten Agenten kommuniziert, der die Begrenzung des Gesamtmassenstroms berücksichtigt. Falls die Summe der gewünschten Massenströme größer als der Gesamtmassenstrom ist, werden alle gewünschten Massenströme mit dem gleichen Faktor skaliert, so dass die Begrenzung des Gesamtmassenstroms eingehalten wird.

6.6.2 Alternating Direction Method of Multipliers

Zuerst wird in diesem Abschnitt der grundlegende Mechanismus des ADMM-Algorithmus erklärt. In Abschnitt 6.6.7 wird die Implementierung in der *AgentLib* und die Kommunikationsstruktur erläutert.

Bei einem Optimierungsproblem werden die Optimierungsvariablen, oder auch primale Variablen genannt, bezüglich einer Kostenfunktion minimiert. ADMM ist dagegen eine duale Methode zur Verteilten Optimierung. Dual bedeutet hierbei, dass die primalen Variablen von jedem Agenten frei gewählt werden können, und stattdessen die dualen Variablen, die durch die koppelnden Nebenbedingungen entstehen, iterativ angepasst werden. ADMM wird häufig im Bereich des Machine Learning angewendet, um große Optimierungsprobleme parallelisiert zu lösen. Da ein Austausch

von Modellinformationen für ADMM nicht erforderlich ist, bietet sie sich allerdings auch für eine Agentenbasierte Regelung an. Umfassende Literatur zur Funktionsweise und Anwendung von ADMM finden sich beispielsweise in Boyd [2010]. Im Folgenden werden kurz die grundlegenden Ideen und Konzepte von ADMM vorgestellt.

Für die agentenbasierte Regelung wurde ein Konsens-ADMM-Algorithmus implementiert. Probleme, die mit Konsens-ADMM gelöst werden können, sind dadurch charakterisiert, dass mehrere Teilsysteme sich auf einen Wert für dieselbe Variable einigen müssen. In der verteilten Regelung eines Gebäudeenergiesystems entspräche dies beispielsweise dem Fall, dass ein RLT-Agent und ein Raumkomfort-Agent sich auf einen Massenstrom aus der Lüftungsanlage einigen müssen.

Um die Funktionsweise von ADMM verständlich zu machen, wird der Algorithmus an einem einfachen unbeschränkten Optimierungsproblem erläutert. Ein solches Problem, das mit Konsens-ADMM gelöst werden kann, hat die Struktur

$$\text{minimize } \sum_{i=1}^{N} f_i(x_i) \tag{6.2}$$

$$\text{s.t. } x_i - z = 0 \tag{6.3}$$

wobei N die Anzahl der teilnehmenden Agenten, x_i die lokale Optimierungsvariable eines Agenten darstellt und z der globale Wert der Optimierungsvariable ist. In ADMM wird die erweiterte Lagrange-Funktion optimiert, die sich aus den notwendigen Optimalitätsbedingungen für (6.3) plus einem Strafterm zusammensetzt. Diese bestimmt sich für (6.3) mit

$$L_\rho(x_1, ..., x_N, z, y) = \sum_{i=1}^{N} \left(f_i(x_i) + y_i^T (x_i - z) + (\rho/2) \left\| x_i - z \right\|_2^2 \right), \tag{6.4}$$

wobei y_i die Lagrange-Multiplikatoren des Agenten i sind und $\rho > 0$ den sogenannten Straffaktor darstellt. Die Lösung des Optimierungsproblems (6.3) befindet sich am Sattelpunkt des augmentierten Lagrangefunktionals (6.3). Dieser kann mit Hilfe einer iterativen Prozedur bestimmt werden, bei der abwechselnd die primalen und die dualen Variablen neu berechnet werden. Diese Prozedur wird im folgenden erläutert, und findet sich zusammengefasst in Algorithmus 1.

Im ersten Schritt bestimmen alle Agenten den optimalen Wert ihrer lokalen Variable x_i am Minimum ihrer Kostenfunktion abhängig der initialen Werte für die globale Variable z und der Lagrange-Multiplikatoren y_i.

$$x_i^{k+1} := \arg\min_{x_i} \left(f_i(x_i) + y_i^{k\,T} (x_i - z^k) + (\rho/2) \left\| x_i - z^k \right\|_2^2 \right) \tag{6.5}$$

Dann wird der neue Wert für die globale Variable bestimmt durch Mitteln aller lokalen Variablen der Agenten.

$$z^{k+1} := \frac{1}{N} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^k \right) \tag{6.6}$$

Das Update der Lagrange-Multiplikatoren der Agenten erfolgt dann über die Differenz zwischen dem Wert der lokalen Variable und der globalen Variable.

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - z^{k+1}) \tag{6.7}$$

Mit den neuen Lagrange-Multiplikatoren werden die Schritte wiederholt, so lange bis ein Konvergenzkriterium erfüllt ist.

Algorithmus 1 Verteilte Optimierung mit ADMM

1: Setze y^0, z^0
2: **do**
3: Jeder Agent $i \in N$: Lösen des lokalen Problems:

$$x_i^{k+1} := \arg\min_{x_i} \left(f_i(x_i) + y_i^{k\,T}(x_i - z^k) + (\rho/2) \left\| x_i - z^k \right\|_2^2 \right) \tag{6.8}$$

4: Berechne den neuen Wert der globalen Variable:

$$z^{k+1} := \frac{1}{N} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^k \right) \tag{6.9}$$

5: Jeder Agent $i \in N$: Aktualisierung des Lagrange-Multiplikators:

$$y_i^{k+1} := y_i^k + \rho(x_i^{k+1} - z^{k+1}) \tag{6.10}$$

6: Setze $k \leftarrow k + 1$
7: **while** nicht konvergiert oder $k \leq k_{\max}$

6.6.3 Sensitivitätsbasierte modellprädiktive Regelung

Ein weiterer Ansatz zur verteilten Regelung ist die sogenannte *sensitivitätsbasierte modellprädiktive Regelung* (SENSI). Hierbei handelt es sich im Kontrast zu ADMM um einen primalen Ansatz. Im Folgenden werden die grundlegenden Ideen und Konzepte dieser Methode vorgestellt, welche zum Beispiel in Scheu and Marquardt [2011] nachzulesen sind.
Ausgangspunkt ist das lineare kontinuierliche System

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}\boldsymbol{u}(t), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{6.11}$$

mit der lokalen Repräsentation

$$\dot{\boldsymbol{x}}_i = \sum_{j=1}^{N} \boldsymbol{A}_{ij}\boldsymbol{x}_j(t) + \boldsymbol{B}_{ij}\boldsymbol{u}_j(t), \quad \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0}, \quad \forall\, i \in \{1, \ldots, N\}\,. \tag{6.12}$$

Das Kostenfunktional des Optimalsteuerungsproblems ist in diesem Fall durch

$$\min_{\boldsymbol{u}} \quad \sum_{i=1}^{N} \int_0^T \boldsymbol{x}_i^\top \boldsymbol{Q}_i \boldsymbol{x}_i + \boldsymbol{u}_i^\top \boldsymbol{R}_i \boldsymbol{u}_i \,\mathrm{d}t \tag{6.13a}$$

$$\text{u.B.v.} \quad \dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{u}, \quad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{6.13b}$$

$$\boldsymbol{D} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} + \boldsymbol{e} \geq 0 \tag{6.13c}$$

definiert. Zunächst wird das Optimierungsproblem durch *Volldiskretisierung* in das quadratische Programm

$$\min_{\boldsymbol{p}} \quad \sum_{i=1}^{N} \Phi_i(\boldsymbol{p}) = \sum_{i=1}^{N} \frac{1}{2} \boldsymbol{p}^\top \boldsymbol{A}^i \boldsymbol{p} + \boldsymbol{p}^\top \boldsymbol{B}^i + \boldsymbol{C}^i \tag{6.14a}$$

$$\text{u.B.v.} \quad \boldsymbol{c}_i(\boldsymbol{p}) = {\boldsymbol{D}^i}^\top \boldsymbol{p} + \boldsymbol{E}^i \geq 0, \quad \forall\, i \in \{1, \ldots, N\} \tag{6.14b}$$

überführt. In diesem wird nunmehr bezüglich des Parametervektors $\boldsymbol{p} = \begin{bmatrix} \boldsymbol{u}^\top(t_0) & \cdots & \boldsymbol{u}^\top(t_{N-1}) \end{bmatrix}$, der sich aus den diskretisierten Stellgrößen über den Optimierungshorizont zusammensetzt, optimiert. Die Schreibweise von (6.14) setzt sich bereits aus mehreren (verkoppelten) Teilproblemen zusammen. Jeder Agent, soll nun eines dieser Teilprobleme lösen, wobei der i-te Agent die Kostenfunktion Ψ_i bezüglich dem i-ten Teil des Parametervektors $\boldsymbol{p}_i$ minimiert, d.h.

$$\min_{\boldsymbol{p}_i} \quad \Phi_i(\boldsymbol{p}) = \frac{1}{2} \boldsymbol{p}^\top \boldsymbol{A}^i \boldsymbol{p} + \boldsymbol{p}^\top \boldsymbol{B}^i + \boldsymbol{C}^i \tag{6.15a}$$

$$\text{u.B.v.} \quad \boldsymbol{c}_i(\boldsymbol{p}) = {\boldsymbol{D}^i}^\top \boldsymbol{p} + \boldsymbol{E}^i \geq 0. \tag{6.15b}$$

Diese sogenannten lokalen Probleme hängen jedoch weiterhin vom gesamten Parametervektor $\boldsymbol{p}$ ab. Zu diesem Zweck wird während der lokalen Optimierung der adaptierte Parametervektor $\tilde{\boldsymbol{p}}_i^{[k]} = \begin{bmatrix} \boldsymbol{p}_1^{[k]}, \ldots, \boldsymbol{p}_{i-1}^{[k]}, \boldsymbol{p}_i, \boldsymbol{p}_{i+1}^{[k]}, \ldots, \boldsymbol{p}_N^{[k]} \end{bmatrix}$ verwendet. Darin wird jede Optimierungsvariable, die nicht dem jeweiligen lokalen Problem zugeordnet ist, konstant gehalten. Der Superindex $[k]$ gibt an, dass für die jeweilige Variable das Ergebnis aus dem k-ten Iterationsschritt eingesetzt wird. Diese werden in jeder Iteration zwischen den Agenten ausgetauscht. Um Konvergenz der lokalen Lösungen gegen die zentrale Lösung zu gewährleisten, werden sogenannten *Sensitivitäten* eingeführt. Sie sind ein Maß für die Änderung der Kostenfunktion eines anderen Agenten im Bezug auf den Parametervektor. Sie werden durch den Gradienten

$$\left.\frac{\partial \Phi_j}{\partial \boldsymbol{p}_i}\right|_{\boldsymbol{p}^{[k]}} - \boldsymbol{\lambda}_j^{[k]} \left.\frac{\partial \boldsymbol{c}_j}{\partial \boldsymbol{p}_i}\right|_{\boldsymbol{p}^{[k]}} \tag{6.16}$$

definiert und stellen somit eine lineare Approximation der Kostenänderung da. Die lokale Kostenfunktion (6.15) wird durch die Sensitivitäten bezüglich aller anderen Agenten erweitert. Die Sensitivitäten müssen wiederum in jeder Iteration zwischen den Agenten ausgetauscht werden. Zusam-

mengefasst ergibt sich somit der Algorithmus 2. Hierbei werden in jeder Iteration die Trajektorien und Sensitivitäten zwischen den Agenten ausgetauscht, bevor das lokale Problem (parallel) gelöst wird. Die Kostenfunktion wird in diesem Kontext noch um den sogenannten Konvexifizierungsterm $(\boldsymbol{p}_i - \boldsymbol{p}_i^{[k]})^\top \boldsymbol{\Omega}(\boldsymbol{p}_i - \boldsymbol{p}_i^{[k]})$ erweitert.

Algorithmus 2 Sensitivitätsbasierte verteilte Optimierung

1: Initialisierung des QP (6.15), shiften von Trajektorien
2: Setze $k = 0$
3: **do**
4: Jeder Agent $i \in N$: Austausch von Trajektorien & Sensitivitäten
5: Jeder Agent $i \in N$: Lösen des lokalen Problems:

$$\min_{\boldsymbol{p}_i} \quad \Phi_i^*(\tilde{\boldsymbol{p}}_i^{[k]}) = \Phi_i(\tilde{\boldsymbol{p}}_i^{[k]}) + \left(\left.\frac{\partial \Phi_j}{\partial \boldsymbol{p}_i}\right|_{\boldsymbol{p}^{[k]}} - \boldsymbol{\lambda}_j^{[k]} \left.\frac{\partial \boldsymbol{c}_j}{\partial \boldsymbol{p}_i}\right|_{\boldsymbol{p}^{[k]}} \right)^\top (\boldsymbol{p}_i - \boldsymbol{p}_i^{[k]}) \tag{6.17a}$$

$$+ (\boldsymbol{p}_i - \boldsymbol{p}_i^{[k]})^\top \boldsymbol{\Omega}(\boldsymbol{p}_i - \boldsymbol{p}_i^{[k]})$$

$$\text{u.B.v.} \quad \boldsymbol{c}_i(\boldsymbol{p}) = \boldsymbol{D}^{i\top} \tilde{\boldsymbol{p}}_i^{[k]} + \boldsymbol{E}^i \geq 0. \tag{6.17b}$$

6: Setze $k \leftarrow k + 1$
7: **while** nicht konvergiert oder $k \leq k_{\max}$

6.6.4 Verhandlungsbasierte modellprädiktive Regelung

Als letzten Ansatz zur verteilten Regelung betrachten wir die sogenannte verhandlungsbasierte modellprädiktive Regelung [Maestre et al., 2011]. Im Folgenden werden die wesentlichen Ideen dieser Methode vorgestellt.

Die Methode geht von mehreren linearen System aus, die lediglich über den Eingang gekoppelt sind

$$x_i(t+1) = A_i x_i(t) + \sum_{j \in n_i} B_{ij} u_j(t).$$

Hierbei wird keine direkte Zuordnung der Eingänge zu den Subsystemen angenommen. Dies bedeutet insbesondere, dass die Anzahl der Eingangscluster sich von der Anzahl der Subsysteme unterscheiden kann. Weiterhin werden lineare, entkoppelte Beschränkungen in den Zuständen und Eingängen für die Subsysteme bzw. Eingangscluster angenommen. Das Ziel ist alle Subsysteme zum Ursprung zu regeln und dabei die Summe über quadratische lokale Kostenfunktionen

$$J_i(x_i, \{U_j\}_{j \in n_i}) = \sum_{k_0}^{N-1} L_i(x_{i,k}, \{u_{j,k}\}_{j \in n_i}) + F_i(x_{i,N})$$

zu minimieren.

Der Ansatz geht davon aus, dass es einen initale Satz von Eingangstrajektorien gibt. Die grundlegende Idee des Ansatzes ist, dass die Agenten nacheinander basierend auf dem lokalen Modell und

der lokalen Kostenfunktionen einen neuen Satz von Eingangstrajektorien vorschlagen. Hierbei werden sinnvollerweise nur die Eingänge eingepasst, welche das Subsystem des Agenten beeinflussen. Die vorgeschlagenen Eingangstrajektorien werden an alle davon beeinflusste Agenten kommuniziert. Diese Agenten berechnen die Änderung in der lokalen Kostenfunktion, welche dann an den vorschlagenden Agenten kommuniziert wird. Dieser Agent wertet die Änderung der globalen Kostenfunktion durch seinen Vorschlag aus. Falls die globalen Kosten durch seinen Vorschlag abnehmen, wird der Satz von Eingangstrajektorien aktualisiert, indem die neuen Eingangstrajektorien an alle Agenten kommuniziert werden. Falls die globalen Kosten steigen, wird der Vorschlag verworfen und ein anderer Agent darf einen neuen Vorschlag berechnen. Dies führt zu dem in Algorithmus 3 dargestellten Ablauf.

Algorithmus 3 Verhandlungsbasierte verteilte Optimierung

1: Initialisierung des Satzes von Eingangstrajektorien $\{U_j^d\}_{j=1,\dots,M_u}$
2: Setze $k = 0$
3: **do**
4: Auswahl eines Agenten i
5: Agent i: Berechnen und Kommunikation neuer Eingangstrajektorien $\{U_j^p\}_{j\in n_i}$
6: Agenten l, dessen Subsystem von $\{U_j^p\}_{j\in n_i}$ abhängig: Berechnen von

$$\Delta J_l^p = J_l(x_l, \{U_j^p\}_{j\in n_l}) - J_l(x_l, \{U_j^d\}_{j\in n_l})$$

Falls Beschränkungen verletzt, setze $\Delta J_l = \infty$. Kommunikation von ΔJ_l^p and Agent i
7: Agent i: Berechnen von

$$\Delta J^p = \sum_l \Delta J_l^p$$

Falls $\Delta J^p < 0$: Aktualisiere und kommuniziere $\{U_j^d\}_{j=1,\dots,M_u}$.
8: Setze $k \leftarrow k + 1$
9: **while** $k \leq k_{\max}$
10: Wende den ersten Teil der Eingangstrajektorien an
11: Agenten $i \in M_x$: Messen des aktuellen Zustands
12: Agenten $i \in M_x$: Kommuniziere $K_{ji}x_{i,N}$ für die Erweiterung der Eingangstrajektorien entsprechend der Formel $\sum_p K_{jp}x_{p,N}$

Der wesentliche Vorteil der verhandlungsbasierten Methode ist der geringe Austausch von Informationen zwischen den Agenten. Insbesondere wird kein Modell der anderen Subsysteme benötigt. Dies stellt für die Akzeptanz im Feld ein wichtiges Kriterium dar. Allerdings können die Agenten dadurch nur das lokale Wissen nutzen, um eine bessere Eingangstrajektorie vorzuschlagen, was sich vermutlich in einer geringeren Performance äußert.

6.6.5 Vergleich der Algorithmen

Im Folgenden werden die verteilten Algorithmen an einem Simulationsbeispiel erprobt. In Anlehnung an den Versuchsaufbau an der RWTH Aachen werden hierbei vier Räume betrachtet. Die

Dynamik eines einzelnen Raums ist durch

$$\dot{T}_i = \frac{1}{C_i}\left(c_p \dot{m}_i (T^{in} - T_i) + d_i\right), \quad T_i(0) = T_{0,i} \tag{6.18}$$

Es werden die folgenden Parameter verwendet:

$$\begin{aligned} T^{in} &= 290\,\mathrm{K} & C_i &= 60\,\mathrm{kJ\,K^{-1}} \\ T_{0,1} &= 296\,\mathrm{K} & d_1 &= 150\,\mathrm{W} \\ T_{0,2} &= 298\,\mathrm{K} & d_2 &= 100\,\mathrm{W} \\ T_{0,3} &= 301\,\mathrm{K} & d_3 &= 50\,\mathrm{W} \\ T_{0,4} &= 303\,\mathrm{K} & d_4 &= 10\,\mathrm{W} \end{aligned}$$

Die Dynamik kann durch Veränderung des Massenstroms beeinflusst werden, womit im regelungstechnischen Sinne $\dot{m}_i = u_i$ gilt. Ziel der Regelung ist, die Temperatur der Räume auf $T_s = 296\,\mathrm{K}$ zu bringen. Jedoch soll hierbei ein Gesamtmassenstrom von $0.1\,\mathrm{kg\,s^{-1}}$ nicht überschritten werden, sprich

$$\sum_{i=1}^{4} u_i \leq 0.1\,\mathrm{kg\,s^{-1}}. \tag{6.19}$$

Somit sind die Agenten dazu gezwungen einen Konsens in Bezug auf die Aufteilung des Massenstroms zu finden.

Hierarchische verteilte modellprädiktive Regelung

Die Simulationsergebnisse der hierarchischen verteilten modellprädiktiven Regelung, angewendet auf das Vierraumbeispiel sind in Abbildung 6.23 dargestellt. Der Ansatz regelt die geforderte Temperatur von 296 K ein. Die Skalierung der gewünschten Luftmassenströme proportional zum gewünschten Gesamtmassenstrom führt dazu, dass Räume mit einem größeren Kühlbedarf aufgrund einer höheren Raumtemperatur oder Wärmeeintrag einen entsprechend größeren Teil des Gesamtmassenstroms zugewiesen bekommen. Dies führt zu dem wünschenswerten Verhalten, dass zu Beginn die Temperatur aller vier Räume sich schnell annähert, was aufgrund der Begrenzung des Gesamtmassenstroms zu einer Erhöhung in Raum 1 führt.

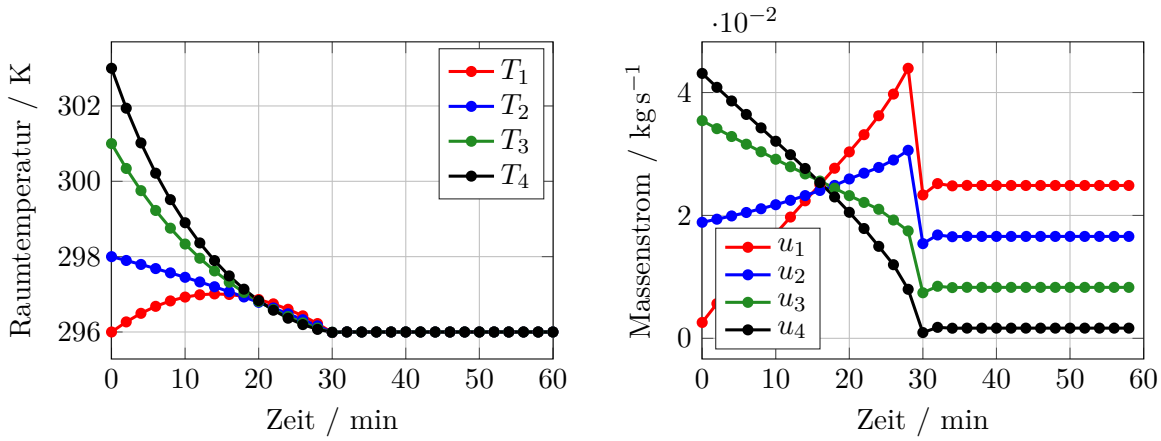


Abbildung 6.23: Simulationsergebnisse der hierarchischen DMPC für das Vierraumbeispiel. Auf der linken Seite sind die Temperaturverläufe und auf der rechten Seite die zugehörigen Massenströme zu sehen.

Verteilte modellprädiktive Regelung mit Alternating Direction Method of Multipliers

Der in der *AgentLib* implementierte ADMM-Algorithmus ist in der Lage, Konsens zwischen geteilten Variablen herzustellen, unterstützt in der jetzigen Form jedoch keine globalen Beschränkungen. Daher wurde zur Implementierung des 4-Raum Beispiels ein fünfter Agent eingeführt, der die RLT-Anlage repräsentiert und die Limitationen des Gesamtmassenstroms kennt.

Die Simulationsergebnisse der ADMM-basierten verteilten modellprädiktiven Regelung angewendet auf das Vierraumbeispiel sind in Abbildung 6.24 dargestellt. Der Verlauf der Temperaturen ist sehr ähnlich verglichen mit der sensitivitätsbasierten modellprädiktiven Regelung, deren Ergebniss im folgenden Abschnitt vorgestellt werden. Es ist zu erkennen, dass der Agent 1 zuerst eine Verschlechterung seines Komforts in Kauf nimmt, um den anderen Agenten zu helfen in ihren Komfortbereich zurückzukehren.

Sensitivitätsbasierte verteilte modellprädiktive Regelung

Die Simulationsergebnisse der sensitivitätsbasierten verteilten modellprädiktiven Regelung, angewendet auf das Vierraumbeispiel, sind in Abbildung 6.25 dargestellt. Es ist zu sehen, dass es in allen Räumen gelingt die geforderte Temperatur von 296 K einzuregeln. Die leichte Abweichung ist auf den Kompromiss zwischen der Bestrafung von Stellgrößen und Zustandsabweichungen zurückzuführen. Außerdem kann man dem Verlauf der Stellgrößen entnehmen, dass die Beschränkung des maximalen Gesamtmassenstroms eingehalten wird. Die Agenten agieren trotz lokaler Optimierung gemäß dem gesamten Regelziel, was besonders am Anfang der Simulation deutlich wird: Hier nimmt Agent 1 eine Verschlechterung seiner Temperatur in Kauf, damit die jeweils anderen Agenten einen größeren Massenstrom zur Verfügung haben.

In Tabelle 6.2 werden der SENSI Algorithmus und ADMM verglichen. Dabei sind einmal die Wur-

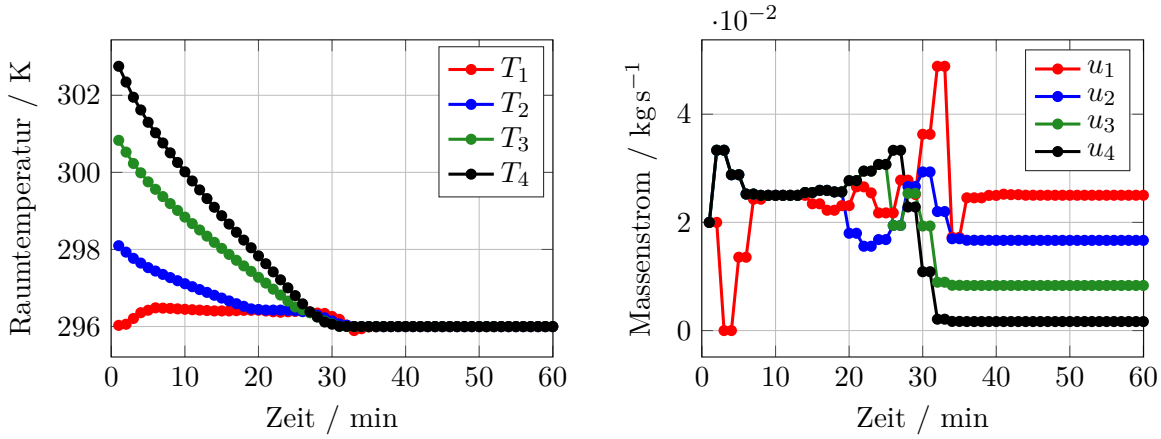


Abbildung 6.24: Simulationsergebnisse des ADMM-basierten DMPC für das Vierraumbeispiel. Auf der linken Seite sind die Temperaturverläufe und auf der rechten Seite die zugehörigen Massenströme zu sehen.

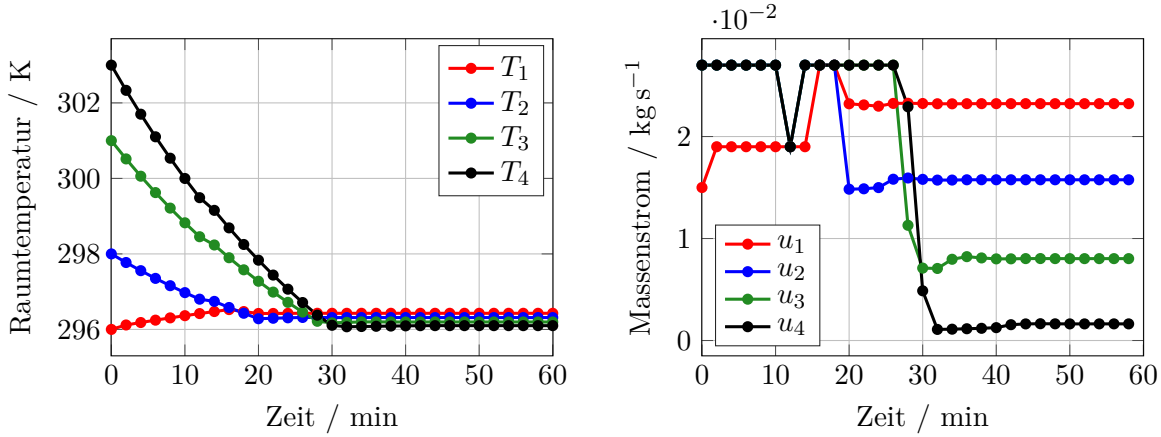


Abbildung 6.25: Simulationsergebnisse des sensitivitätsbasierten DMPC für das Vierraumbeispiel. Auf der linken Seite sind die Temperaturverläufe und auf der rechten Seite die zugehörigen Massenströme zu sehen.

zel des mittleren quadratischen Fehlers (Root mean squared error - RMSE) und der integrierte absolute Fehler (integrated absolute error - IAE) der Raumtemperatur aller Räume über einen Simulationszeitraum von einer Stunde aufgetragen.

Tabelle 6.2: Vergleich der Algorithmen

	Sensi	ADMM
RMSE	1.628 K	1.384 K
IAE	3.365 Kh	2.798 Kh

Dabei ist ersichtlich, dass beide Algorithmen eine ähnlich gute Regelqualität aufweisen, wobei beim ADMM eine leicht geringere Abweichung festzustellen ist.

6.6.6 Erweiterung der Sensitivitätsbasierten modellprädiktiven Regelung

Die bisherige Formulierung des SENSI Algorithmus hat drei große Nachteile:

- Es sind nur lineare Systeme möglich
- Die Struktur wird durch eine Volldiskretisierung verschleiert
- Es wird die zentrale Dynamik mitgerechnet.

Dennoch verspricht der SENSI Algorithmus Vorteile gegenüber ADMM. Ein großer Vorteil ist der geringere Kommunikationsaufwand, da nur zwei Kommunikationsschritte anstelle von drei vonnöten sind. Dieser kann in einem realen Agentensystem nicht vernachlässigbar sein. Ein weiterer Punkt stellt die Realisierbarkeit von suboptimalen Lösungen dar. Der Grund hier liegt an dem primalen Ansatz vom SENSI.

Aus diesen Gründen wurde die Idee des SENSI Algorithmus auf allgemeine, nichtlineare Optimalsteuerungsprobleme erweitert. Hier soll der Grundgedanke präsentiert werden, genaueres ist Huber and Graichen [2021] zu entnehmen.

Hauptaugenmerk der dynamischen Formulierung ist zunächst die Sensitivität. Im statischen Fall war diese die partielle Ableitung der Kostenfunktion nach den Koppelgrößen. Das Äquivalent im dynamischen Fall ist die Gâteaux-Ableitung des Kostenfunktionals. Daraus folgt

$$\delta J_j(\boldsymbol{u}_i; \delta \boldsymbol{u}_i) = \int_0^T \left(\partial_{\boldsymbol{u}} H_j[t]\right)^T \delta \boldsymbol{u}_i(t) \mathrm{d}t \tag{6.20}$$

mit dem Index i für die lokalen Größen, dem Index j für nicht lokale Größen und der Hamiltonfunktion

$$H_j[t] = l_j(\boldsymbol{x}_j, \boldsymbol{u}_j) + \boldsymbol{\lambda}_j^T \boldsymbol{f}(\boldsymbol{x}_j, \boldsymbol{x}_{-j}^{[k]}, \boldsymbol{u}_j, \boldsymbol{u}_{-j}^{[k]}). \tag{6.21}$$

Es folgt für das lokale Optimalsteuerungsproblem

$$\min_{\boldsymbol{u}_i} V_i(\boldsymbol{x}_i(T)) + \int_0^T l_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \mathrm{d}t \tag{6.22}$$

$$+ \sum_j \delta J_j(\boldsymbol{u}_i^{[k]}; \boldsymbol{u}_i - \boldsymbol{u}_i^{[k]}) \tag{6.23}$$

$$u.b.v.\ \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{x}_{-i}^{[k]}, \boldsymbol{u}_i, \boldsymbol{u}_{-i}^{[k]}), \quad \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \tag{6.24}$$

$$\boldsymbol{u}_i \in [\boldsymbol{u}_i^-, \boldsymbol{u}_i^+]. \tag{6.25}$$

Um die Leistungsfähigkeit zu testen, wurde der SENSI-Algorithmus in C++ implementiert. Die lokalen Probleme werden mit der Toolbox GRAMPC Englert et al. [2018] gelöst. Als Vergleich wurde der ADMM-Algorithmus in der Toolbox GRAMPC-D Burk et al. [2020] herangezogen, da diese in C++ implementiert wurde.

Der ausführliche Vergleich ist in Huber et al. [2022] dargestellt. Es hat sich gezeigt, dass die Rechenzeitskalierung der beiden Algorithmen ähnlich ist. Ein vielversprechendes Ergebnis ist die Konver-

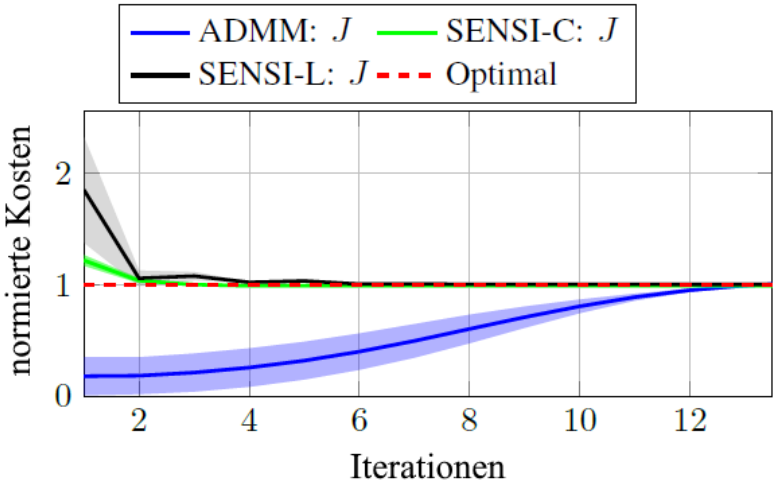


Abbildung 6.26: Konvergenzverhalten von SENSI und ADMM. Rot stellt die optimalen Kosten dar.

genz: SENSI konvergiert deutlich schneller als ADMM, wie in Abbildung 6.26 zu sehen ist. Anhand des Konvergenzverhaltens lässt sich zusätzlich sagen, dass Suboptimale Lösungen von SENSI immer realisierbar sind. Ein Problem von SENSI ist, diese Konvergenz zu erreichen. Daher ist im derzeitigen Zustand ADMM robuster. Die Verbesserung des SENSI-Algorithmus ist weiterhin aktueller Gegenstand der Forschung.

6.6.7 Implementierung des ADMM in der AgentLib

Um die verteilten Algorithmen in das Agentenframework zu integrieren und sie für die Demonstration nutzbar zu machen, wurde das *AgentLib_MPC*-Plugin erweitert. Basierend auf den Untersuchungen der Algorithmen und der Literatur, wurde der ADMM Algorithmus für eine Implementierung gewählt, da er flexibler und allgemeingültiger ist als der hierarchische Ansatz, und stabiler und besser erforscht als der SENSI-Ansatz.

ADMM-Modul

Der ADMM-Algorithmus wurde in der *AgentLib* implementiert, bestehend auf dem entwickelten *MPC*-Modul und dem *CasadiBackend*. Die grundlegenden Mechanismen und ein simulativer Vergleich des ADMM-Algorithmus wurden bereits in Abschnitt 6.6.2 und 6.6.5 erläutert. Hier wird die Implementierung in der *AgentLib* und die Kommunikationsstruktur erklärt. Anschließend werden Simulationsergebnisse für ADMM an einem nichtlinearen Energiesystem vorgestellt.

Eine besondere Herausforderung bei der realen Implementierung des ADMM-Algorithmus stellt die Synchronisation der Agenten dar. Bei einer zentralen MPC löst der MPC-Agent in regelmäßigen Abständen das Optimierungsproblem und aktualisiert den Wert der Steuerungsvariable. Damit in

einem verteilten System ein Konsens über ADMM erzielt und korrekte Steuerungssignale gesendet werden können, müssen die ADMM-Agenten ihre lokalen Optimierungen zeitgleich durchführen.

Es wurden zwei verschiedene Strukturen für ADMM-basierte MPC entwickelt, um dieses Problem zu lösen. Eine Version, in der ein Koordinator existiert, der mit allen Teilnehmeragenten kommuniziert und bestimmt, wann ein Regelschritt gestartet wird, wie die ADMM-Iterationen innerhalb der Optimierung ablaufen, und wann die Iterationen abgebrochen werden. Eine andere Variante stellt ein vollständig dezentrales System dar, in dem die Agenten sich an der Zeit synchronisieren.

Zunächst soll die dezentrale Variante vorgestellt werden. Hier orientieren sich alle ADMM-Agenten an der realen Uhrzeit, um ihre Regelschritte zu annähernd gleichen Zeitschritten durchzuführen.
Als Herausforderung kommt die Anforderung an die Kommunikationsstruktur, dass sie flexibel ist im Bezug auf das Hinzufügen und Entfernen bzw. Ausfallen von Agenten während des Betriebs. Um dieses Problem zu lösen, wurde eine Registrierungsstruktur eingeführt, bei der vor jeder Optimierung alle teilnehmenden Agenten bestätigt werden. So wird die Teilnahme neuer Agenten ermöglicht und vermieden, dass auf die Ergebnisse nicht mehr verfügbarer Agenten gewartet wird. Ein Agent registriert sich, indem er vor Beginn eines neuen Regelschrittes den initialen Wert seiner lokalen Variable sendet.
Der detaillierte Prozessablauf eines einzelnen Agenten ist in Algorithmus 4 dargestellt.

Algorithmus 4 Prozessablauf eines ADMM-Agenten

```
Warte auf gemeinsamen Start der Optimierung (fester Zeitplan)
do
    Sende lokale Variablen.
    Warte die Registrierungsdauer ab.
    Registriere alle verfügbaren Agenten.
    Initialisiere Lagrange-Multiplikatoren und Globalvariable.
    do
        Löse lokales Optimalsteuerungsproblem.
        Sende lokale Variablen.
        Warte auf alle registrierten Teilnehmer.
        Aktualisiere Globalvariablen und Lagrange-Multiplikatoren.
    while Nicht konvergiert, oder Iterationslimit überschritten.
    Sende optimale Werte der Steuerungsvariable an Aktor.
    Warte bis zum nächsten gemeinsamen Start der Optimierung.
while Wahr
```

Koordinator-Agent

Die Implementierung vieler verteilter Optimierungsalgorithmen beruht auf einem zentralen Koordinator, der die teilnehmenden Agenten kennt, und zwischen ihnen vermittelt. Um *AgentLib_MPC* auf verschiedene verteilte Algorithmen vorzubereiten, wurde eine Koordinatorstruktur als Basisklasse erstellt, und darauf der ADMM mit Koordinator aufgebaut.

Der Koordinator erleichtert das Zusammenspiel zwischen den Agenten und erlaubt die Berechnung von sinnvollen Konvergenzkriterien. Der Koordinator wickelt zum einen die Registrierung von Agenten, die an der Optimierung teilnehmen ab. Zum Anderen, ist er dafür verantwortlich, dass Agenten, die nicht mehr reagieren oder erreichbar sind (timeout), beim Durchführen von synchronen Regelalgorithmen nicht die Ausführung des gesamten Algorithmus blockieren. Die Koordinatorfunktion könnte auch von einem zufällig ausgewählten Agenten übernommen werden, wird aber aus pragmatischen und systematischen Gründen im Falle von *AgentLib_MPC* als eigenständiger Agent implementiert.

Um die Berechnungsschritte eines verteilten Algorithmus synchronisieren zu können, muss der Koordinator darüber informiert sein, welche Agenten an der Optimierung teilnehmen. Dies wird über einen Registrierungsprozess zwischen Koordinator und Agent abgewickelt. Das zugehörige Sequenzdiagramm ist in Abbildung 6.27 dargestellt. Der Agent startet die Registrierung, indem er eine Registrierungsanfrage an den Koordinator sendet. Dieser führt zunächst eine temporäre Registrierung durch, was bedeutet, dass die Informationen des Agenten gespeichert werden und zusätzlich ein Status für den jeweiligen Agenten angelegt wird. Dieser beschreibt den Zustand des Agenten, der vorerst auf *pending* gestellt wird, was soviel heißt, dass der Registrierungsprozess noch nicht abgeschlossen ist. Anschließend schickt der Koordinator relevante Optimierungsparameter (wie z. B. Abtastzeit, Horizontlänge, etc.) an den Agenten. Die Nachricht wird gleichzeitig als Antwort auf die Registierungsanfrage des Agenten interpretiert. Erhält dieser die Nachricht, führt er mit Hilfe der übermittelten Informationen eine Initialisierung durch. Sobald diese abgeschlossen ist, ist der Agent bereit für die Optimierung und teilt dies dem Koordinator durch das Senden einer *ready*-Nachricht mit. Der Koordinator schließt dann die Registrierung ab, und setzt entsprechend den Status des

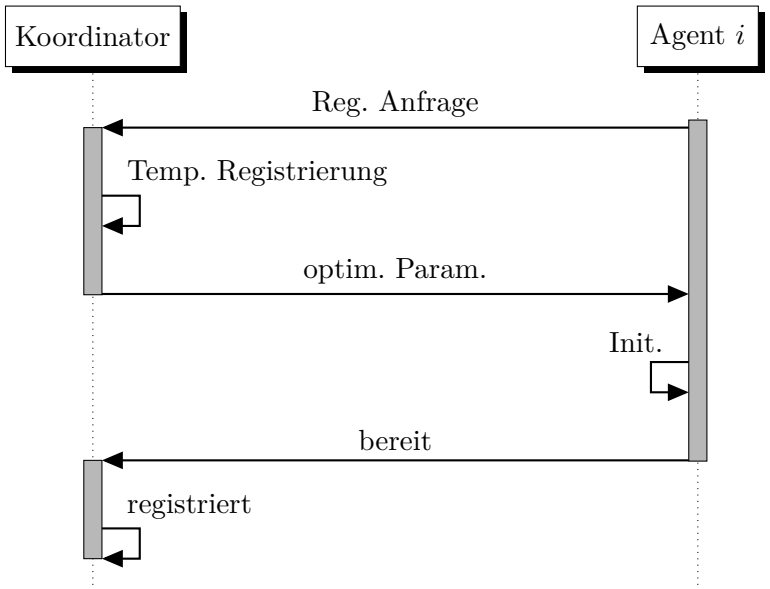


Abbildung 6.27: Sequenzdiagramm des Registrierungsprozesses zwischen einem beliebigen Agenten und dem Koordinator.

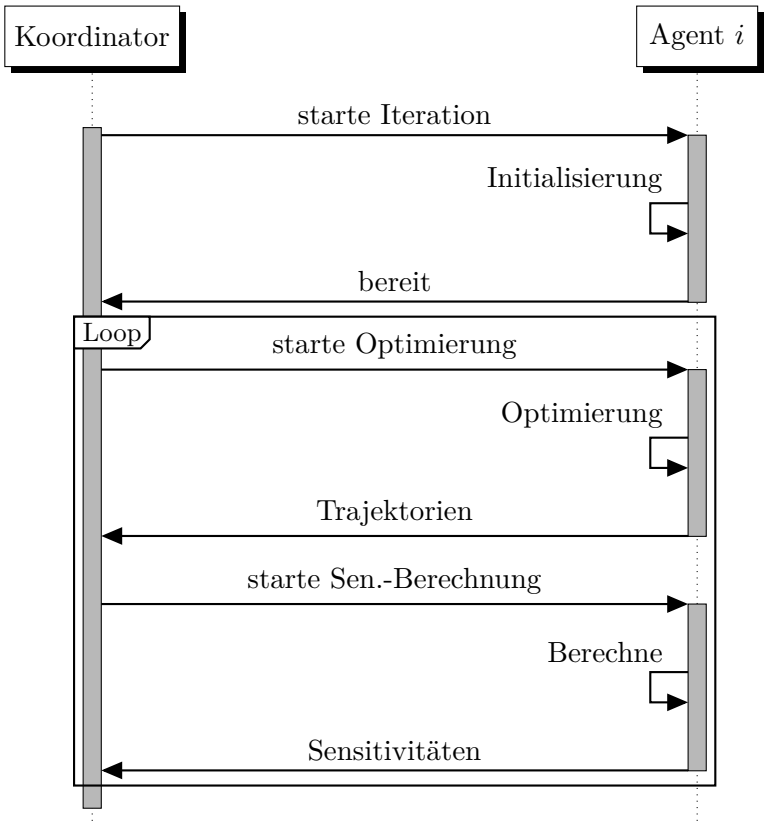


Abbildung 6.28: Sequenzdiagramm zwischen einem beliebigen Agenten und dem Koordinator für eine Iteration während der Optimierung mit dem sensitivitätsbasierten Algorithmus.

Agenten auf *ready*.

Die Hauptaufgabe des Koordinators ist, die Algorithmusschritte von verteilten Algorithmen zu synchronisieren. Im Folgenden ist der Ablauf im Bezug auf den sensitivitätsbasierten Ansatz erklärt, die grundsätzliche Funktionsweise ist aber direkt auf andere Algorithmen übertragbar. Der zugehörige Ablauf ist in Abbildung 6.28 dargestellt. In der *AgentLib* kann aufgrund der abstrakten Basisklasse für den Koordinator der Anwender selbstständig die Funktionsweise auf die individuellen Abläufe anpassen. Im ersten Schritt, wird an alle Agenten, die den Status *ready* haben, die Information für das Starten des Algorithmus gesendet. Dadurch hat jeder Agent die Möglichkeit gegebenenfalls notwendige Initialisierungen wie zum Beispiel das Verschieben von Trajektorien oder festlegen von Anfangswerten durchzuführen. Ist dies erfolgt, wird das dem Koordinator wiederum über das senden einer *ready*-Nachricht mitgeteilt. Daraufhin werden vom Koordinator die Algorithmusschritte getriggert. In jedem dieser Schritte wird von den Agenten eine Antwort erwartet, die signalisiert, dass der Berechnungsschritt durchgeführt wurde. Um blockierendes Verhalten zu vermeiden, ist zusätzlich eine maximale Antwortzeit im Koordinator hinterlegt. Wird diese überschritten, so werden alle ausstehenden Agenten als unerreichbar interpretiert und der Algorithmus läuft weiter. Die Schleife über die Algorithmusschritte wird so lange durchlaufen, bis ein Konvergenzkriterium

erreicht ist, oder bis die maximale Anzahl der Iterationen erreicht wurde.

Basierend auf der neuen Koordinatorstruktur wurden für die ADMM-Implementierung ein Koordinatoragent und ein Subagent entwickelt. Der Koordinator ist in dieser Struktur dazu in der Lage, nach jeder Iteration das Primale Residual und das Duale Residual zu bestimmen, um frühzeitig abzubrechen, wenn die Konvergenz erreicht ist. Verglichen mit dem Ansatz einer fixen Iterationszahl kann so die Konvergenz des Algorithmus und somit die Regelgüte gewährleistet werden, während der Kommunikationsaufwand reduziert wird.

Simulationsergebnisse des ADMM mit Koordinator an einem komplexen Energiesystem

Der ADMM-Algorithmus der *AgentLib* wurde an einem komplexeren, nichtlinearen Verbundsystem aus Energieerzeugern und Verbrauchern getestet. Das Beispielsystem ist in Abbildung 6.29 zu sehen. Es besteht aus einem geschichteten Warmwasserspeicher, einem Blockheizkraftwerk, einem Gaskessel und zwei exemplarischen Verbrauchern, die beispielsweise thermische Zonen eines größeren Gebäudes darstellen könnten.

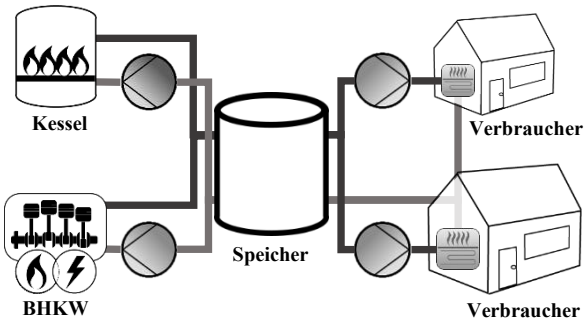


Abbildung 6.29: Beispielgebäudeenergiesystem für die ADMM-Untersuchung

Für die verteilte Regelung wurde das System in drei Regelungsagenten unterteilt - einen für das Erzeugersystem bestehend aus BHKW, Kessel, Warmwasserspeicher und Heizkreislauf, und jeweils einen für die Verbraucher. Die Systemgrenze wird um die Heizkörper gezogen, sodass die Schnittstellengrößen zwischen den Systemen den Wärmeübergang an der Heizfläche darstellen müssen. Da der Wärmeübergang von der Temperaturdifferenz des Raums zum Heizkreislauf abhängt, sind zwei Größen als Schnittstelle nötig, beispielsweise die beiden Temperaturen in Raum und Heizkörper. Alternativ kann nur eine Temperatur und der Wärmestrom als Schnittstellenvariable gewählt werden. Dies bietet den Vorteil, dass das physikalische Wissen über den Wärmeübergang basierend auf der Temperaturdifferenz nur in einem der beiden Systeme vorhanden sein muss. Daher wurden als Schnittstellengröße die Temperatur des Heizkreises T_h und die Temperatur der thermischen Zone T_z gewählt. Die entstehende Struktur der Optimierungsmodelle ist in Abbildung 6.30 zu sehen.

Die Simulationen für das beschriebene System sind mit einem Temperaturprofil aus Aachen, Deutschland, im Februar 2015 durchgeführt worden. In diesem Zeitraum sind die Außentemperaturen ge-

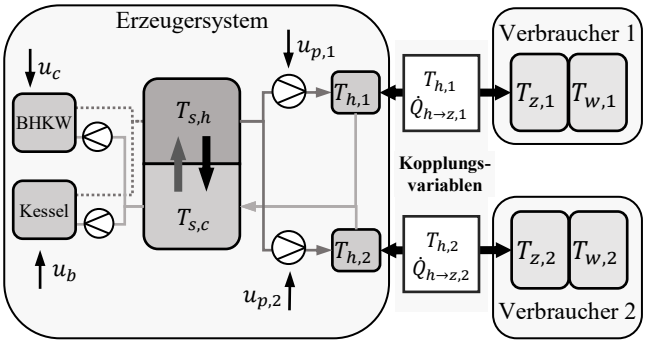


Abbildung 6.30: Aufteilung des Eneriesystems in Subsysteme

ring, sodass, der Regler die Erzeuger teils unter Vollast betreiben muss, um komfortable Raumtemperaturen zu erreichen. Die Simulationen wurden mit einer Abtastzeit von 3600 s und einem Prädiktionshorizont von 12 Schritten durchgeführt, was zu einer Vorhersagelänge von 12 Stunden führt. Die Kostenfunktion wurde mit einem Gaspreis von 0,071 $\frac{€}{kWh}$ und einem Stromverkaufspreis von 0,12 $\frac{€}{kWh}$ gewählt, die dem typischen Stand von 2021 entsprechen. Die Parameter q_i in der Kostenfunktion für die Raumtemperatur sind so gewählt, dass die Kosten für eine Abweichung von 1 K vom Komfortbereich über eine Stunde 20 € und 40 € für die kleine bzw. große thermische Zone entsprechen. Die Simulationen wurden mit dem aggregierten Optimierungsmodell durchgeführt, so dass es keine Diskrepanz zwischen dem Prozess- und dem Anlagenmodell gibt. Der Strafparameter wurde konstant zu $\rho = 1$ gesetzt. Die Trajektorien für die dualen Variablen y und die globale Variable z werden mit der Lösung des vorherigen Zeitschritts initialisiert, die um ein Abtastintervall verschoben wurde. Das letzte Abtastintervall wird wiederholt. Bei der ersten Iteration wird eine Initialisierung nahe des Optimums für die globale Variable durchgeführt, während die dualen Variablen mit null initialisiert werden. Abbildung 6.31 zeigt die Ergebnisse des ADMM-basierten Reglers im geschlossenen Regelkreis, wobei die Toleranzen auf $r_0 = 0.001$ und $s_0 = 0.005$. Wie zu sehen ist, stabilisiert der Regler das System erfolgreich. Ebenso bevorzugt er das BHKW gegenüber dem Kessel zur Wärmeerzeugung. Im Durchschnitt benötigte eine Optimierung 747 Iterationen und 51,85 s Rechenzeit, was noch innerhalb der Echtzeitanforderung ist, da das Abtastintervall 3600 s beträgt.

Abbildung 6.32a zeigt die Konvergenz des ADMM Algorithmus für die erste Optimierung über 60.000 Iterationen. In Abbildung 6.32a zeigt das obere Diagramm die Abnahme der primären und dualen Residuen, die zunächst starke Instabilitäten zeigen, insbesondere beim primalen Residuum. Dennoch erreicht das primale Residuum schnell Werte in der Größenordnung von 10^{-5}, während das duale Residuum nur langsam abnimmt. Die untere Grafik in Abbildung 6.32a zeigt die Entwicklung der Kostenfunktion. Nach etwa 40.000 Iterationen nimmt der Zielwert nicht mehr ab. An einigen Stellen gibt es große Ausschläge des Zielwerts nach unten. Diese sind Fälle, in denen der lokale Optimierer nicht konvergieren konnte. Es wird deutlich, dass eine extreme Anzahl von Iterationen für den ADMM-basierten Regler erforderlich ist, um das Optimalsteuerungsproblem mit hoher Genauigkeit zu lösen. Die langsame Abnahme des dualen Residuums führt auch zu dem

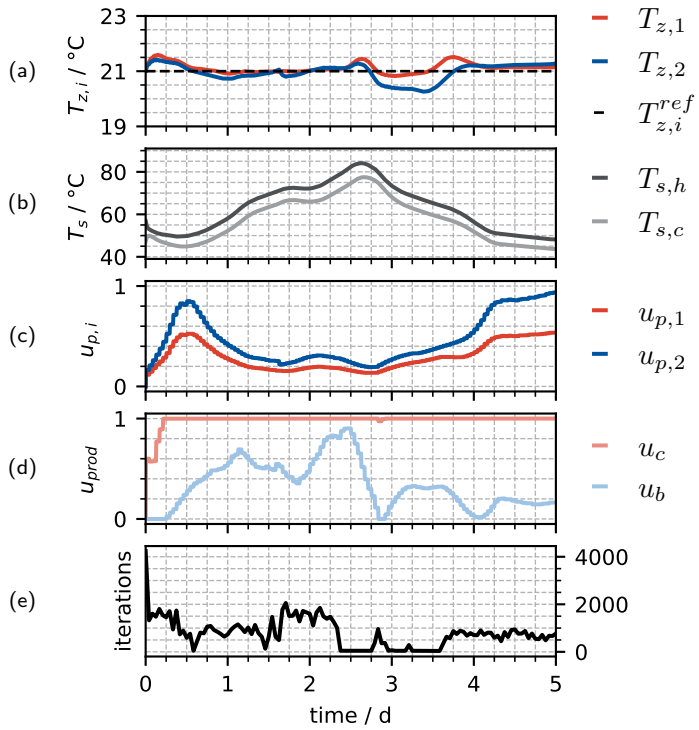
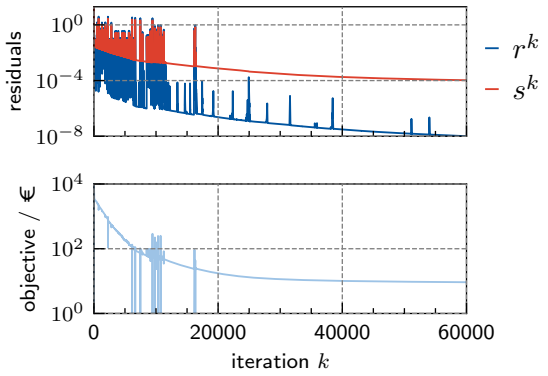


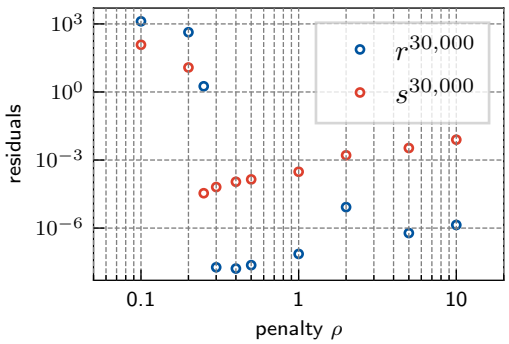
Abbildung 6.31: Ergebnisse der verteilten MPC. Primales und Duales Residuum sind zu $r_0 = 0.001$ und $s_0 = 0.005$ gesetzt. (a) Raumtemperaturen für Verbraucher 1 ($T_{z,1}$) und Verbraucher 2 ($T_{z,2}$), sowie die Referenztemperatur ($T_{z,i}^{ref}$); (b) Speichertemperaturen für die obere Schicht ($T_{s,h}$) und die untere Schicht ($T_{s,c}$); (c) Relative Pumpendrehzahlen für Zone eins und zwei; (d) relative Erzeugerleistung für das BHKW (u_c) und den Kessel (u_b); (e) ADMM-Iterationen

Problem, dass es schwierig ist, ein geeignetes Konvergenzkriterium zu finden, bei dem die Leistung des Reglers im geschlossenen Regelkreis zufriedenstellend ist.

Für eine optimale Konvergenz des ADMM ist die Wahl eines geeigneten Strafparameters ρ entscheidend. Abbildung 6.32b zeigt die Werte der dualen und primären Residuen nach 30.000 Iterationen für verschiedene Werte von ρ. Für kleine Werte wie 0,1 ist der Algorithmus instabil und schwankt um große Residuen. Eine Erhöhung dieses Wertes auf etwa 0,3 führt schließlich zu stabiler und vergleichsweise schneller Konvergenz. Bei einer weiteren Erhöhung von ρ bleibt der Algorithmus stabil, jedoch verlangsamt sich die Konvergenz. Es muss einen optimalen Wert für den Strafparameter um 0,3 herum geben muss. Dennoch haben wir für die Simulationen hier einen Wert von 1 gewählt, um ausreichenden Abstand von der Instabilitätszone zu schaffen. Tabelle 6.3 zeigt die Leistung des Reglers im geschlossenen Regelkreis für verschiedene Werte der dualen Toleranz s_0 und des zentralen Problems. Eine kleinere duale Toleranz verbessert die Genauigkeit des Reglers und verringert den mittleren quadratischen Fehler (RMSE, *root mean squared error*) der Raumtemperaturregelung. Im Gegenzug steigt jedoch die benötigte Iterationszahl und somit auch die Rechenzeit. Die Energiekosten nur geringfügig durch die verringerte Genauigkeit bei größeren To-



(a) Konvergenz des ADMM-Algorithmus.



(b) Konvergenz nach 30.000 Iterationen für verschiedene Werte des Strafparameters ρ.

Tabelle 6.3: Kennwerte des Systems mit geschlossenem Regelkreis über sieben Tage.

Duale Toleranz	RMSE / K	Energiekosten / €	Durchschnittliche Iterationszahl
zentral	0.0044	647.1	-
0.0005	0.028	649.9	4423
0.001	0.056	655.5	2682
0.005	0.254	655.5	747
0.01	0.778	667.1	394

leranzen beeinflusst.

6.7 Prüfstand

Die entwickelten verteilten Algorithmen wurden inklusive der benötigten Infrastruktur anhand von Prüfstandsversuchen getestet (siehe Abbildung 6.33 auf Seite 88).

6.7.1 Infrastruktur und Anbindung

Am Lehrstuhl für Gebäude- und Raumklimatechnik wurde der bestehende Senken-Hardware-in-the-Loop Prüfstand zur Analyse und Bewertung von Heizkörpern und Thermostatventilen im Rahmen des Projektes zu einem Agenten-in-the-Loop (AiL) Prüfstand umgerüstet. Damit wurde die frühzeitige Validierung agentenbasierter Regelungskonzepte an realer Hardware ermöglicht.

Ein typisches Beispiel in der agentenbasierten Betriebsoptimierung ist ein Mehr-Raum-Beispiel. Hierbei sollen sich die Agenten auf z.B. eine Zulufttemperatur oder einen Volumenstrom einigen. Zur experimentellen Abbildung dieses Use-Cases wurde der bestehende Prüfstand um die Möglichkeit der CO_2-Einspritzung sowie der dezentralen Volumenstromregelung erweitert. Die Erweiterung der CO_2-Einspritzung ist in Abbildung 6.33 dargestellt.

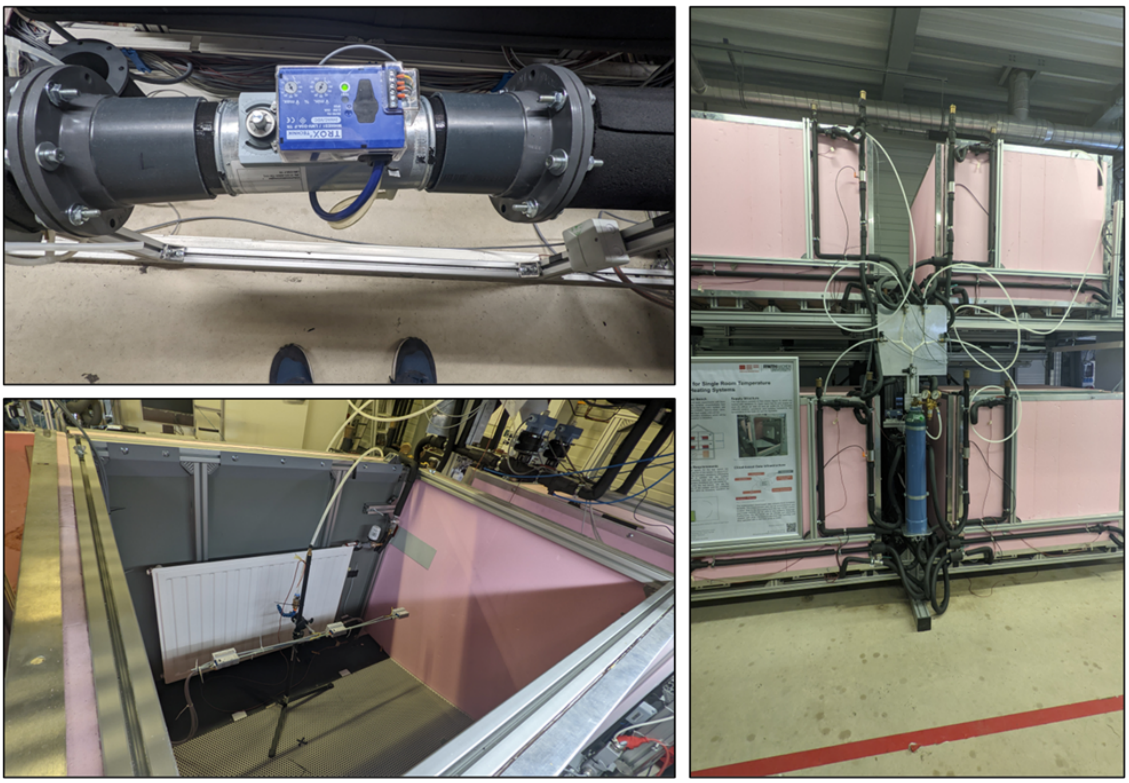


Abbildung 6.33: Erweiterung des Prüfstands. Oben links die dezentrale Volumenstromregelung, unten links die CO_2-Sensoren sowie rechts die CO_2-Einspritzung mitsamt der vier Kammern des Agent-in-the-Loop Prüfstands dargestellt.

Durch die vier Zonen und der zentralen Zuluft können ideal verteilte Agentenansätze validiert werden. Als zentrale Agenten können folgenden Komponenten aufgefasst werden:

- Zuluft: Zulufttemperatur und Volumenstrom
- Wandtemperierung: Vorlauftemperatur und Massenstrom
- Heizkreis: Vorlauftemperatur und Massenstrom

Für jeden der vier Räume können die Agenten beliebig definiert werden. Beispielsweise müssen die Raumagenten mit den zentralen Agenten verhandeln, welcher zentrale Zustand eingestellt wird. Ob Nutzende jeden Raum als Agenten auffassen, oder pro Raum drei Agenten definieren (CO_2-Konzentration, Wandtemperatur, und Raumtemperatur) ist offen.

Die Werte der Experimente werden in einer lokalen Benutzeroberfläche (GUI) und im Webbrowser über Grafana visualisiert. Die Hauptübersicht der GUI ist in Abbildung 6.34 dargestellt. Hier wird ebenso die mögliche Aufteilung in unterschiedliche Agenten ersichtlich. Durch die Anbindung an FIWARE werden semantische Informationen der einzelnen Sensoren und Aktoren zugänglich gemacht. Ebenso werden die Zeitreihendaten über *QuantumLeap* in der *CrateDB* gespeichert.

Weiterhin wurde eine lokale Prüfstandsregelung konzeptioniert, welche einen sichere cloud- und

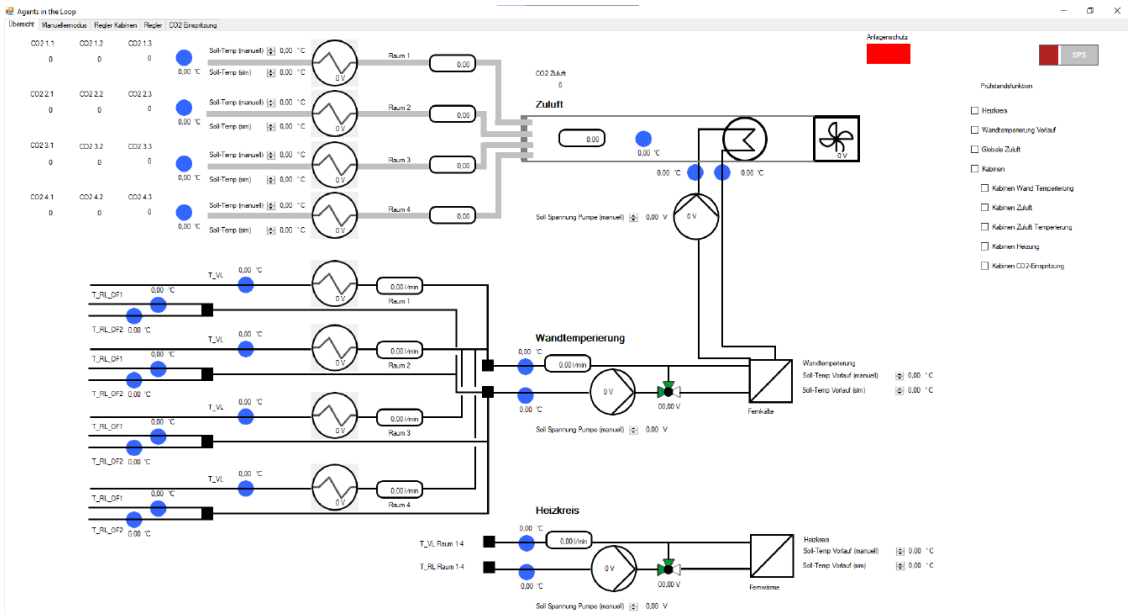


Abbildung 6.34: Benutzeroberfläche des Agent-in-the-Loop Prüfstands

agentenbasierte Regelung ermöglicht. Falls keine cloudbasierten Werte geschrieben werden, übernimmt die lokale Prüfstandsregelung die Bereitstellung des Nutzerkomforts.

6.7.2 Semantisches Modell

In Abbildung 6.35 ist der HIL-Prüfstand schematisch dargestellt. Ausgehend vom Zuluftventilator strömt die Luft durch ein zentrales Kühlregister zu den vier Kabinen. Über elektrische Nacherhitzer lässt sich das Temperaturniveau zusätzlich anheben, die Volumenströme in die Kabinen können einzeln geregelt werden. Zur Belegungssimulation ist eine steuerbare CO_2-Einspritzung vorhanden.

Zur Anbindung des Prüfstands wurde zunächst ein semantisches Modell erstellt, aus dem anschließend möglichst automatisch die Agentenkonfiguration generiert werden sollte. Das Prüfstandsmodell wurde in RDF[4] basierend auf der Brick-Ontologie[5] erstellt und stellt den Prüfstand als Wissensgraph dar. Diese Designentscheidung wurde vor dem Hintergrund getroffen, dass das Agentensystem mit entsprechendem Kommunikationsgraphen ebenfalls eine Graphstruktur aufweist und daher die Konfiguration des Systems über ein graphbasiertes Modell naheliegt.

Abbildung 6.36 zeigt Auszüge des zugehörigen semantischen Modells des Prüfstands. Die Luft strömt entlang der `brick:feedsAir` Relationen bzw. *Object Properties* durch die verschiedenen Komponenten bis hin zu den vier Kabinen, hier als `brick:Room` modelliert. Die Sensorik und Aktorik ist ebenfalls über die entsprechenden Brick-Klassen abgebildet. Die zugehörigen Instanzen enthalten als *Datatype Property* den verwendeten Alias bzw. Datenpunktnamen im Automations-

[4]Resource Description Framework
[5]Brick – A uniform metadata schema for buildings (https://brickschema.org/)

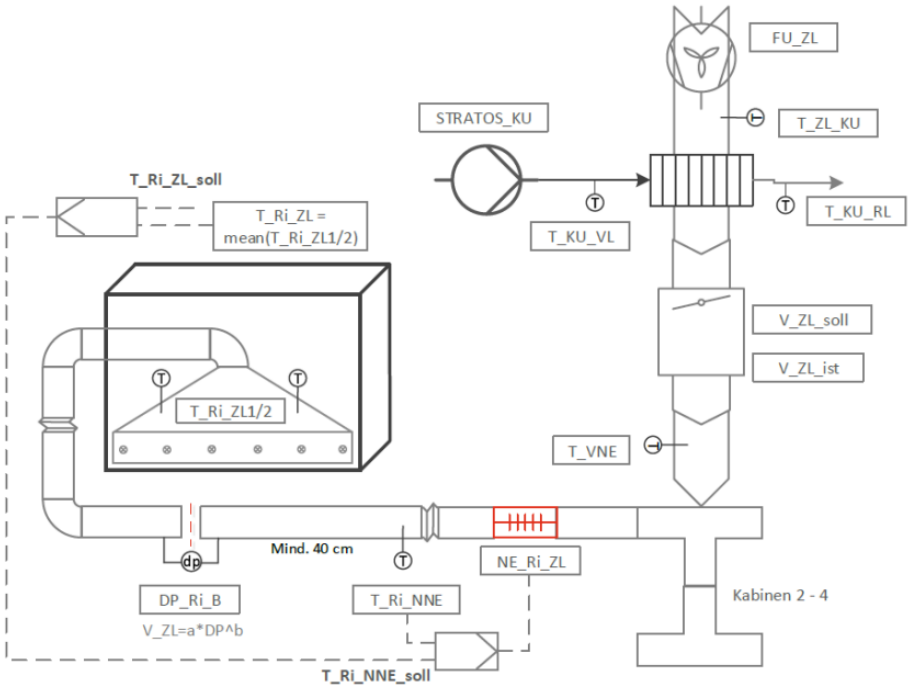


Abbildung 6.35: Schematische Darstellung des Prüfstands

system, so dass mit Hilfe dieser Information die Verbindung zur realen Anlage hergestellt werden kann.

Über die graphenbasierte Abfragesprache SPARQL können entsprechende Inhalte aus dem Modell abgefragt werden, um die flachen Konfigurationsdateien zur Erstellung des Agentensystems zu erzeugen. So können zum Beispiel alle Räume innerhalb des Modells inklusive der Sensorik und Aktorik mit zugehörigen Datenpunktbezeichnungen abgefragt werden, um entsprechende Agenten zu instanziieren, Verbindung mit dem Automationssystem herzustellen und die benachbarten Agenten als Kopplungen innerhalb der verteilten MPC einzutragen.

So kann zum Beispiel die Abfrage aus Auflistung 6.7 genutzt werden, um alle Räume und ihre zugehörigen Datenpunkte, sowie die Art der Datenpunkte zu erhalten.

Auflistung 6.7: Definition einer Variablen

```
qres = g.query(
"""SELECT ?room_label ?volume ?point_type ?point_alias
   WHERE {
       ?room a brick:Room .
       ?room rdfs:label ?room_label .
       ?room brick:volume/brick:value ?volume .
       ?room brick:hasPoint ?point .
       ?point a ?point_type .
```

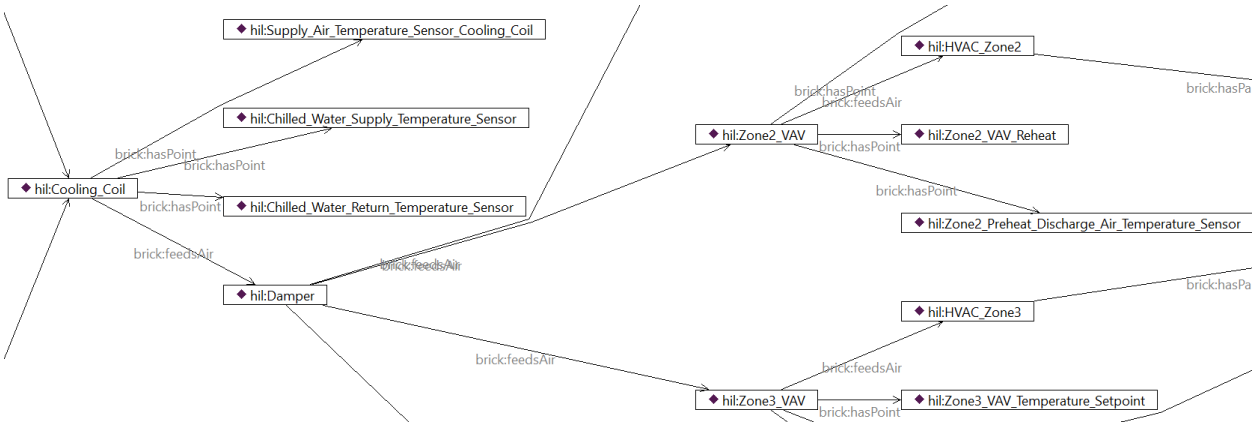


Abbildung 6.36: Semantisches Modell des Prüfstands (Auszug)

```
            ?point hil:hasAlias ?point_alias .
        }"""
    )
```

Mit dieser Information kann für jeden Raum eine Agentenkonfiguration erstellt werden, die die Messwerte und Aktorik der jeweiligen Datenpunkte zur Verfügung hat. Ebenfalls kann die verteilte modellprädiktive Regelung für den Raum basierend auf dieser Information erstellt werden. Da alle Agenten auf dasselbe semantische Modell für ihre Informationen zugreifen, ist die Nomenklatur eindeutig und Schnittstellen können über String-Matching realisiert werden. Innerhalb dieses Projekts ist der Code zur Erstellung der Agenten aus semantischen Modellen noch auf den speziellen Anwendungsfall am Senken-HiL beschränkt. In Zukunft ist jedoch vorstellbar, dass Softwarebibliotheken aufgebaut werden, die typische Anwendungsfälle abdecken und so ganze Agentensysteme aus semantischen Gebäudedaten aufbauen können.

6.7.3 Ergebnisse

Schließlich wurde am Prüfstand ein Versuch durchgeführt, bei dem ein mit der *AgentLib* implementiertes Multi-Agenten-System an den Prüfstand angebunden wurde. Der Versuch wurde über eine Zeit von 30 Minuten durchgeführt. Die CO_2-Einspritzung wurde von einem Belegungsplan gesteuert. Das Agentensystem wurde dabei lokal auf einem Rechner ausgeführt, der mit dem Prüfstandsrechner kommuniziert hat.

Die Regelung der Zonen erfolgte über eine verteilte modellprädiktive Regelung mit ADMM, analog zu den Simulationen aus Abschnitt 6.6. Die einzelnen Zonen sind an einen Zuluftagenten gekoppelt, der die Beschränkung für den maximalen Luftvolumenstrom gewährleisten soll. Die modellprädiktive Regelung wurde mit einer Abtastzeit von 120 s und einem Prädiktionshorizont von 4 durchgeführt. Der Strafparameter der ADMM war 0.3 und die maximale Iterationszahl wurde auf 70 beschränkt.

Das Agentensystem konnte mit dem HiL-Prüfstand verbunden werden und hat die Volumenströme für die einzelnen Zonen vorgegeben. Die Ergebnisse des Versuchs sind in Abbildung 6.37 zu sehen. Die zentralen Beschränkungen von $550\,\mathrm{m^3\,h^{-1}}$ wurden eingehalten. Es zeigt sich eine Diskrepanz zwischen Soll- und Ist-Wert, der durch den Messaufbau erklärbar ist. Die Ergebnisse der einzelnen Räume sind in Abbildung 6.38 bis 6.41 dargestellt. Hier kann man sehen wie Schrittweise sich die Vorgabe des Volumenstroms ändert und der Ist-Wert diesem auch folgt. Der CO_2-Grenzwert kann bei allen Räumen nicht eingehalten werden. Dafür gibt es zwei Ursachen. Zum einen war der CO_2-Eintrag pro Raum sehr groß und nur durch Pulsweitenmodulation einzustellen, sodass die Anforderung insgesamt groß waren. Auf der anderen Seite war die Ergebnisqualität durch die langsame Konvergenz des ADMM-Algorithmus begrenzt. Im Rahmen der Versuchsdurchführung mussten Anpassungen im Modell und der Parametrierung vorgenommen werden, um mit dem Prüfstand kompatibel zu sein. Da der ADMM-Algorithmus sensibel gegenüber Änderungen der Parameter ist, konnte während des Versuchs nicht die optimale Regelgüte erreicht werden. Um diese Probleme in Zukunft zu vermeiden, können automatische Skalierungsverfahren implementiert werden.

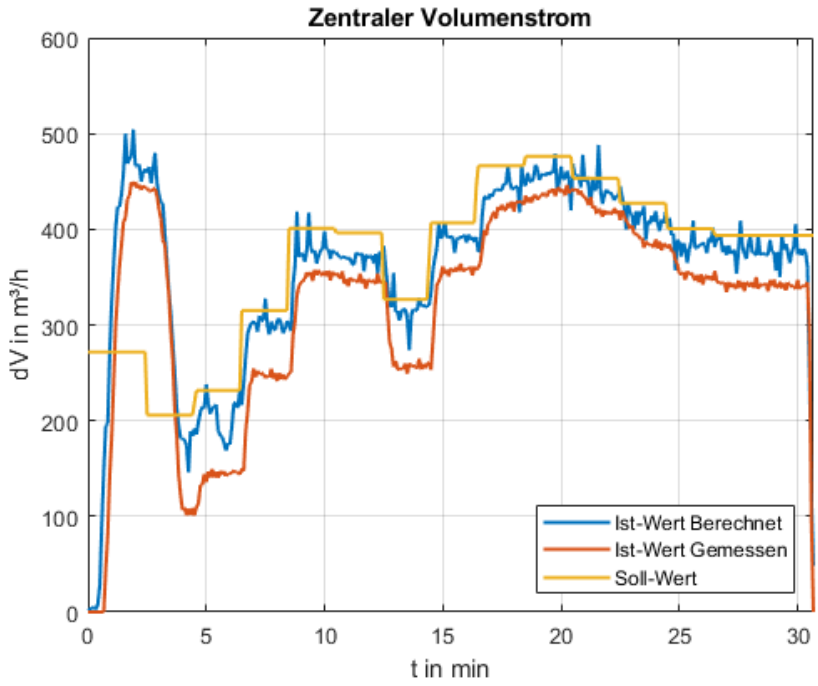


Abbildung 6.37: Zentraler Volumenstrom im HiL Versuch

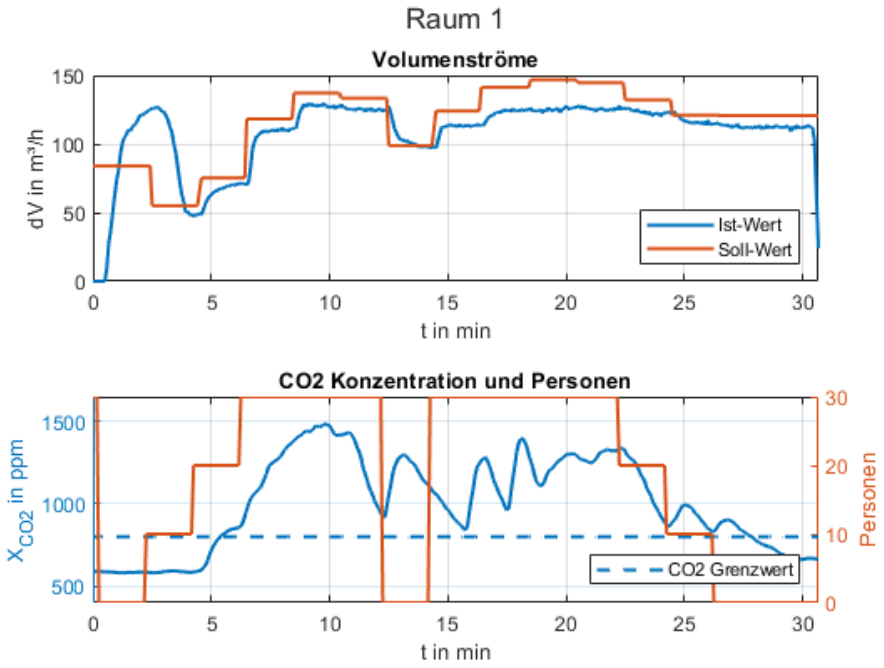


Abbildung 6.38: Volumenstrom und CO_2-Konzentration von Raum 1

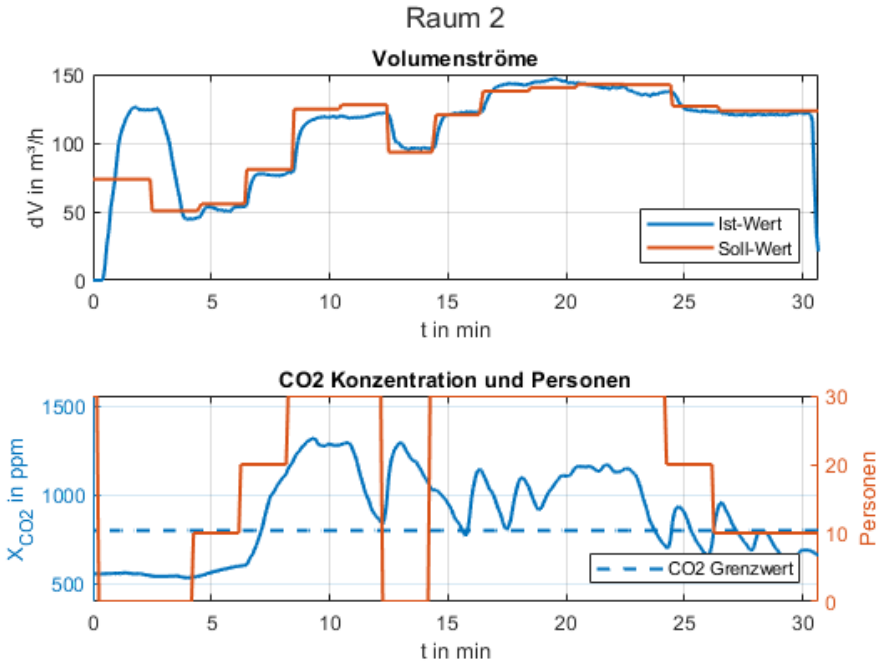


Abbildung 6.39: Volumenstrom und CO_2-Konzentration von Raum 2

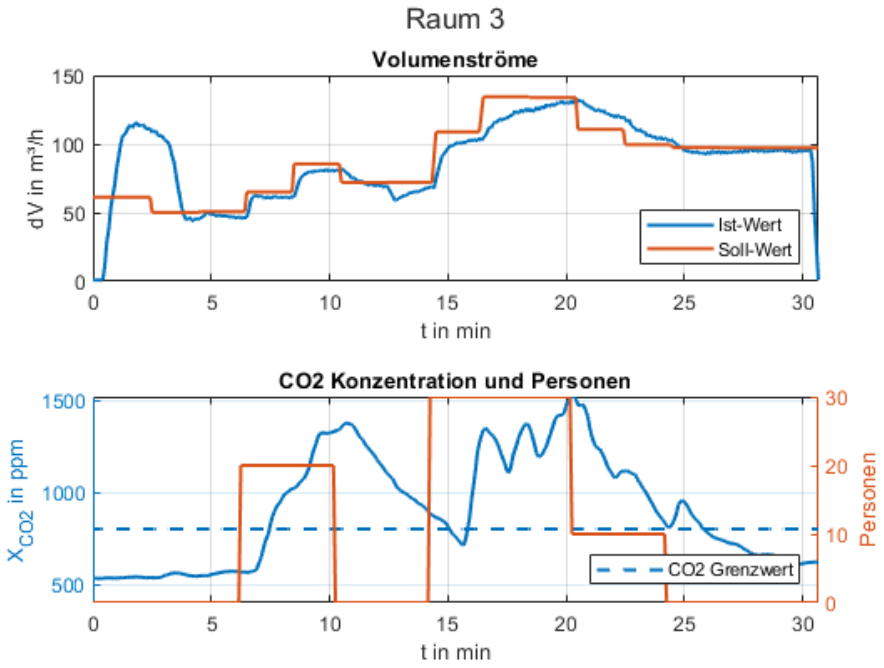


Abbildung 6.40: Volumenstrom und CO_2-Konzentration von Raum 3

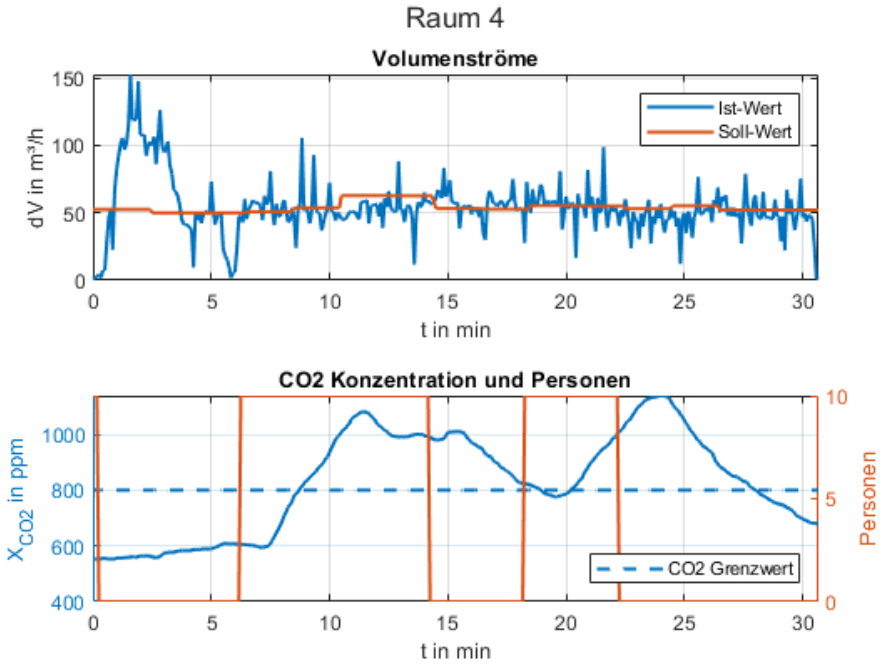


Abbildung 6.41: Volumenstrom und CO_2-Konzentration von Raum 4

6.8 Demonstrator

Die in den vorigen Kapiteln vorgestellten Methoden und Verfahren wurden neben einer simulativen Evaluation auch praktisch angewendet. Neben dem Prüfstand an der RWTH Aachen (siehe Abschnitt 6.7) wurde dazu ein Gebäude am Forschungscampus Renningen der Robert Bosch GmbH genutzt.

6.8.1 Aufbau und Infrastruktur

Als Demonstrator dient das Gebäude Rng111 der Robert Bosch GmbH in Renningen. Dieses Gebäude beinhaltet einen Bürobereich mit Großraumflächen, Kommunikationszonen und Besprechungsräumen verschiedener Größe. Die technische Gebäudeausrüstung umfasst zentrale Lüftungsanlagen (WRG, Heizen, Kühlen, Entfeuchten), Heizung (Radiatoren, teilweise hydraulische Nacherhitzer) sowie Kühldecken in größeren Besprechungsräumen und Außenverschattung.

Im Rahmen vorheriger Projekte wurde ein Bereich von 650 m^2 der ersten Etage mit zusätzlicher Sensorik und Aktorik nachgerüstet, um als Demonstrator für verschiedene Forschungsvorhaben zu dienen.

Die folgenden Komponenten wurden dabei in die bestehende Gebäudeautomation integriert:

- Stellantriebe an den Lüftungsklappen, um den Volumenstrom an nahezu jedem Luftauslass separat einstellen zu können,
- Funkthermostate an den Heizkörpern,
- Kombisensoren (Temperatur, Feuchte, CO_2, VOC) zur Luftqualitätsmessung in der Abluft der verschiedenen Zonen,
- Wärmemengenmessung an Heiz- und Kühlregister sowie statischer Heizung,
- Differenzdrucksensoren im Kanalsystem zur Schlechtpunktüberwachung,
- Fensterkontakte,
- Globe-Thermometer zur Messung der Strahlungstemperatur.

Abbildung 6.42 zeigt den Grundriss innerhalb des Gebäudes, der für die Demonstration genutzt wird, inklusive der Platzierung der verschiedenen Sensoren.

In Abbildung 6.43 ist das Bedienbild der entsprechenden Lüftungsanlage zur Versorgung der Büroflächen des Gebäudes und damit insbesondere der Demonstrationsfläche dargestellt. Die Umgebungsluft strömt durch den Rotationswärmeübertrager, den Zuluftventilator, Kühl- und Heizregister bevor sie in die Zone eintritt. Die Register werden über ein Nahwärmenetz am Campus mit Wärme und Kälte versorgt. Die Lüfterleistung wird auf einen Differenzdruck von 52 Pa am Schlechtpunkt im Kanalsystem, dem Punkt mit dem höchsten Druckverlust, geregelt.

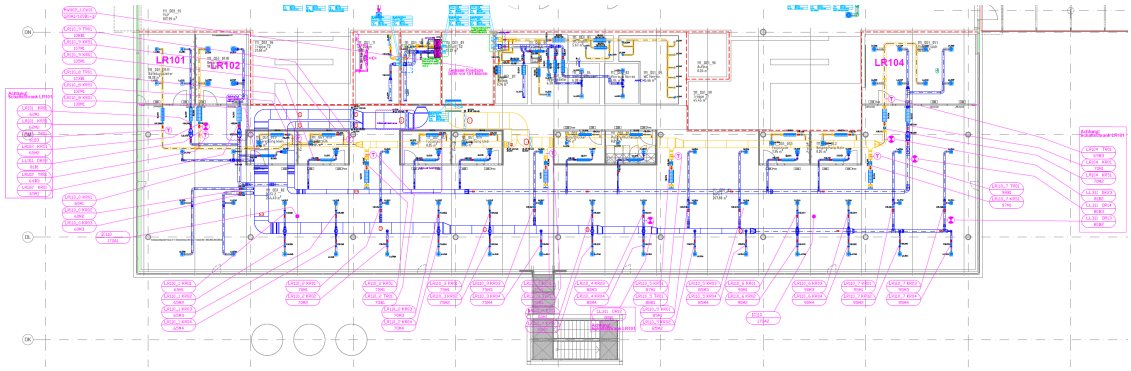


Abbildung 6.42: Sensorik und Aktorik in Rng111/1

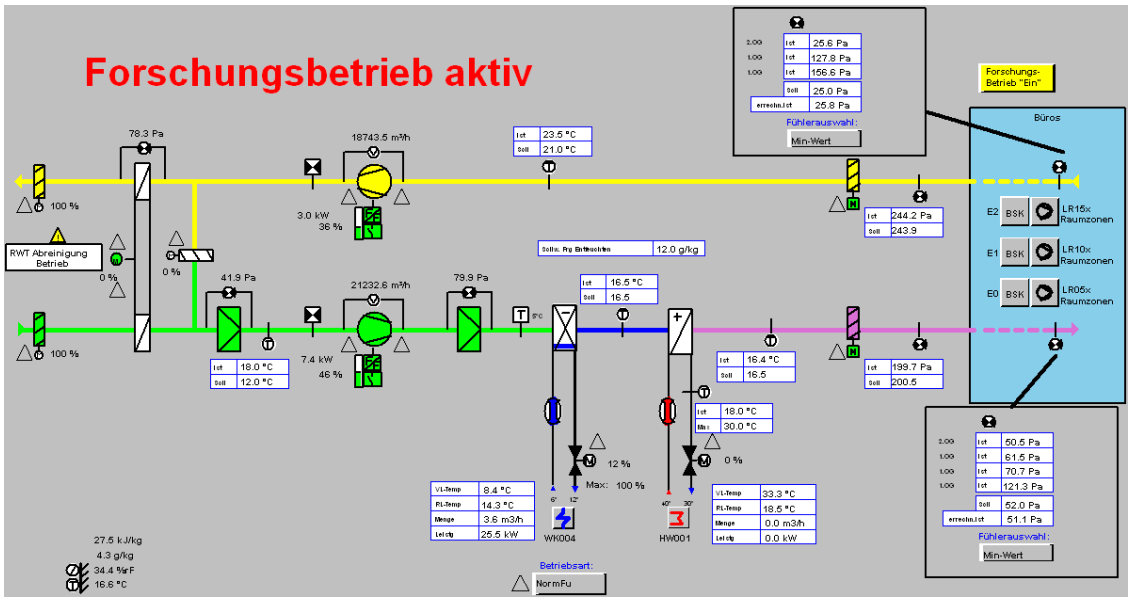


Abbildung 6.43: Bedienbild der Lüftungsanlage LL311

Abbildung 6.44: Grafana-Dashboard

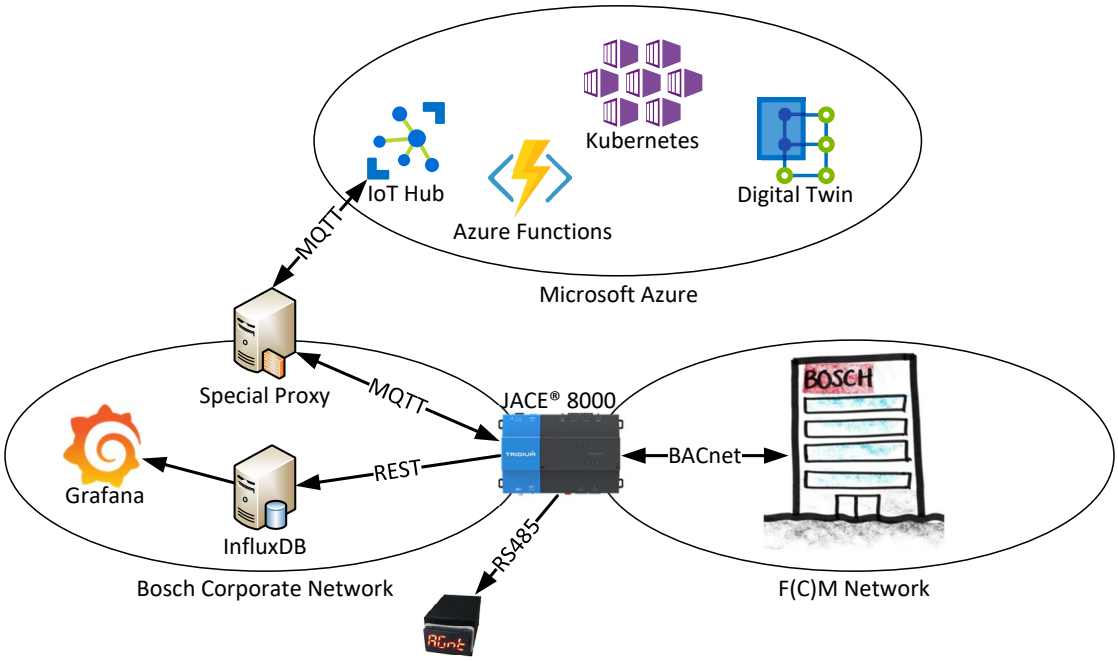


Abbildung 6.45: Backend-Infrastruktur mit Cloud-Anbindung

Alle Komponenten der technischen Gebäudeausrüstung sind über BACnet/IP an die Gebäudeautomation angebunden und werden in einer zentralen Zeitreihendatenbank historisiert. Zur Visualisierung wird Grafana[6] verwendet. In Abbildung 6.44 ist ein Ausschnitt eines entsprechenden Dashboards dargestellt.

Zur prototypischen Umsetzung von neuen Konzepten ist ein JACE 8000[7] Controller mit der BACnet-Domäne verbunden, um Algorithmen basierend auf dem Niagara-Framework zu entwickeln. Damit ist es möglich, beliebige Datenpunkte zu lesen und zu schreiben. Dieser Controller wird genutzt, um die Cloud-Anbindung des Gebäudes zu realisieren.

In Abbildung 6.45 ist die Backend-Infrastruktur dargestellt. Die Verbindung zur Gebäudetechnik wird über den JACE hergestellt. Dieser schreibt relevante Informationen ebenfalls in die Zeitreihendatenbank, um sie über Grafana visualisieren zu können. Zusätzlich werden Datenpunkte als Telemetrie über MQTT an einen IoT Hub in Microsoft Azure übertragen, der über *Commands* auch schreibend auf Datenpunkte zugreifen kann. Damit ist eine bidirektionale Cloud-Anbindung des Gebäudes hergestellt.

6.8.2 Automatische Erstellung eines semantischen Gebäudemodells

Die Erstellung des semantischen Modells erfolgt automatisch aus den Telemetriedaten des Gebäudes. Ein einzelner Datenpunkt besteht dabei nicht nur aus Namen und Wert, sondern enthält als

[6] https://grafana.com/
[7] https://www.tridium.com/de-de/products-services/niagara4#JACE8000

JSON-Objekt zusätzlich eine Menge von *Tags*, die diesem Datenpunkt hinzugefügt wurden. In Auflistung 6.8 ist beispielhaft ein einzelner Messwert der Raumtemperatur im Raum A.160 dargestellt.

Auflistung 6.8: Beispiel eines Datenpunkts (LR102TR01)

```
origin: jace8000
payload:
  Asset:
    DatapointName: LR102TR01
    DeviceName: '126093'
    Instance: '583'
    ObjectType: AnalogInput
    PresentValue: '22.94'
    Tags:
      agent:zoneName: A.160
      hs:air: M
      hs:cur: M
      hs:equipRef: A.160
      hs:id: h:5b232
      hs:kind: Number
      hs:maxVal: '80.0'
      hs:minVal: -inf
      hs:point: M
      hs:sensor: M
      hs:temp: M
      hs:tz: Berlin
      hs:unit: "celsius;\xB0C;(K);+273.15;"
      hs:zone: M
      hs:zoneAirTempSensor: M
      n:displayName: LR102TR01
      n:input: M
      n:name: LR102TR01
      n:ordInSession: station:|h:5b232
      n:point: M
      n:station: JACE
      n:type: control:NumericPoint
  GatewayName: JACE
  Timestamp: '2021-07-14T17:56:46.346+02:00'
```

Der Datenpunkt enthält neben den BACnet-Informationen eine Menge von *Tags* aus verschiedenen Namensräumen. Tags mit dem Prefix `n:` enthalten interne Informationen aus dem verwendeten Niagara-Framework[8], das auf der Steuerung verwendet wird. Das `hs:`-Prefix bezeichnet Haystack-Tags[9], die für das automatische Onboarding, d. h. Erstellung eines digitalen Zwillings des Gebäudes, verwendet werden.

Projekt Haystack ist eine Initiative, die semantische Datenmodelle und Webservices im Gebäudebe-

[8] https://www.tridium.com/us/en/Products/niagara
[9] https://project-haystack.org/doc/appendix/tags

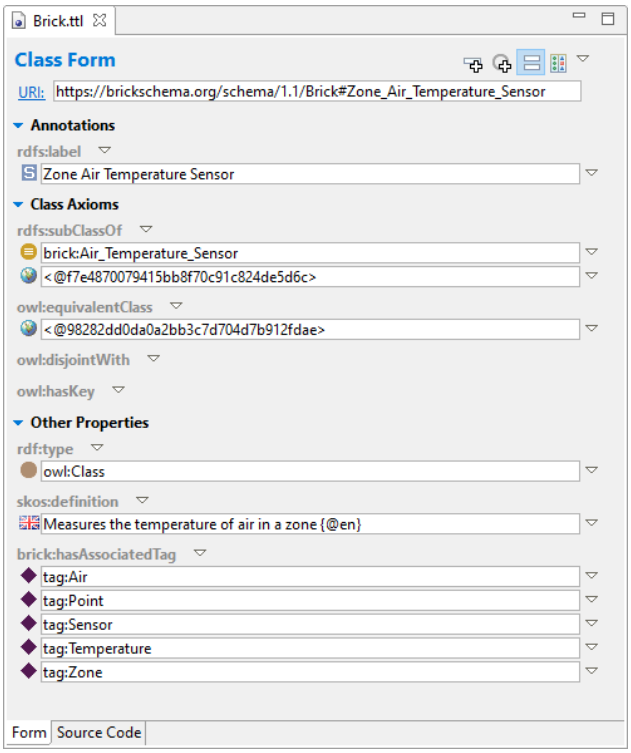


Abbildung 6.46: Definition eines Lufttemperatursensors innerhalb einer Zone in Brick

reich standardisiert. Grundlage des Systems bilden Haystack-Tags, die eine semantische Beschreibung von Entitäten im IoT-Umfeld ermöglichen. Tags sind dabei definierte Schlüssel/Wert-Paare, z. B. `curVal` für den aktuellen Wert eines Datenpunktes. *Marker* sind spezielle Tags ohne Wert, bei denen allein die Anwesenheit eine semantische Bedeutung hat, z. B. wird eine Entität, die einen Raum beschreibt, mit `room` markiert. Im obigen Beispiel ist bei Marker-Tags ein `M` als Wert eingetragen. Ein Vorteil von Haystack ist, dass es in einigen Gebäudeautomationssystemen nativ unterstützt wird und dadurch eventuell bei Bestandsprojekten bereits implementiert ist.

Zur Erstellung des digitalen Zwillings des Gebäudes wird die Brick-Ontologie[10] verwendet. Diese beinhaltet eine Taxonomie von in der Gebäudetechnik relevanten Objekten und basiert auf einem an Haystack angelehnten Tagging-Mechanismus. In Abbildung 6.46 ist die Klassendefinition eines Lufttemperatursensors innerhalb einer Zone in Brick dargestellt. Beim Vergleich der über `brick:hasAssociatedTag` hinzugefügten Tags mit Auflistung 6.8 fällt auf, dass diese überlappen. Dadurch ist es möglich, aus einem Haystack-Objekt eine passende Brick-Klasse zu wählen, indem die Klasse mit der größten gemeinsamen Schnittmenge der Tags bestimmt wird.

Statische Komponenten, wie zum Beispiel Räume als Teil der Gebäudetopologie, emittieren keine Telemetrie, so dass diese nicht durch obigen Mechanismus erfasst werden würden. Trotzdem sind sie Teil der Haystack-Definition (z. B. das `room`-Tag). Damit auch diese Komponenten im digitalen Zwilling erzeugt werden, werden die Informationen explizit in der Steuerung hinterlegt und ebenfalls

[10] https://brickschema.org/

zyklisch als Haystack-Objekt übertragen.

Zusätzlich zu den Datenobjekten definiert Haystack auch Referenzen zu anderen Objekten, wodurch eine Graphstruktur innerhalb des Modells erzeugt wird. Referenzen sind dabei `siteRef`, `ahuRef` oder `equipRef`, um eine Komponente mit einem Ort, einem Lüftungsgerät oder allgemein mit eine Equipment zu verknüpfen. Diese Referenzen werden auf die entsprechenden *Object Properties* `brick:hasLocation` und `brick:isFedBy` abgebildet. Eine Referenz auf ein Equipment wird entweder auf die Relation `brick:isPartOf` oder `brick:isPointOf` abgebildet, je nachdem, ob es sich bei der Zielklasse um einen Datenpunkt handelt oder nicht.

Durch obiges Vorgehen ist es möglich, aus eingehenden Telemetriedaten ein Instanzmodell und damit einen digitalen Zwilling mithilfe der Brick-Ontologie zu erstellen. Um dieses mit ADT zu nutzen, wurde eine Mapping-Ontologie generiert, in der alle Brick-Klassen, die assoziierte Tags enthalten, mit entsprechenden DTDL-Interface-Id's verknüpft werden. In Auflistung 6.9 ist ein Auszug dieses Mappings im Turtle-Format dargestellt. Man sieht, wie der bestehenden `brick:Zone_Air_Temperature_Sensor`-Klasse zusätzliche Triple hinzugefügt werden, um die relevanten Informationen für ADT zu ergänzen.

Auflistung 6.9: Mapping Ontologie

```
brick:Zone_Air_Temperature_Sensor a azureDTDL:Interface ;
    azureDTDL:hasComment "Measures the temperature of air in a zone" ;
    azureDTDL:hasContents <http://bosch.com/cr/BrickMapping#id>,
        <http://bosch.com/cr/BrickMapping#name>,
        <http://bosch.com/cr/BrickMapping#value>,
        brick:hasLocation,
        brick:isFedBy,
        brick:isPartOf,
        brick:isPointOf ;
    azureDTDL:hasDescription "Zone Air Temperature Sensor" ;
    azureDTDL:hasId "dtmi:com:bosch:brick:Zone_Air_Temperature_Sensor;1" .
```

Aus diesem Mapping kann die DTDL-Beschreibung automatisch generiert und damit zur Laufzeit die entsprechenden Modelle instanziiert werden. In Auflistung 6.10 ist das entsprechende Modell des Temperatursensors in DTDL für ADT dargestellt.

Auflistung 6.10: Konvertierte DTDL-Definition des Lufttemperatursensors

```
{
    "@context": "dtmi:dtdl:context;2",
    "@id": "dtmi:com:bosch:brick:Zone_Air_Temperature_Sensor;1",
    "@type": "Interface",
    "comment": "Measures the temperature of air in a zone",
    "contents": [
        {
            "@type": "Relationship",
```

```
            "name": "isPointOf"
        },
        {
            "@type": "Property",
            "description": "The id.",
            "name": "id",
            "schema": "string"
        },
        {
            "@type": "Property",
            "description": "The name.",
            "name": "name",
            "schema": "string"
        },
        {
            "@type": "Relationship",
            "comment": "Subject is physically located in the location given by the object",
            "name": "hasLocation"
        },
        {
            "@type": "Relationship",
            "name": "isPartOf"
        },
        {
            "@type": "Property",
            "description": "The value.",
            "name": "value",
            "schema": "float"
        },
        {
            "@type": "Relationship",
            "name": "isFedBy"
        }
    ],
    "description": "Zone Air Temperature Sensor"
}
```


Dadurch ist es möglich, bei jedem eingehenden Datenpaket einen entsprechenden Twin in ADT zu erzeugen oder bei einem bereits existierenden Twin ein Update des aktuellen Werts durchzuführen. Nach und nach entsteht dabei automatisch ein vollständiges Gebäudemodell in ADT. Ein Ausschnitt dieses Modells ist in Abbildung 6.47 dargestellt.

Instanziierung oder Update von Twins erzeugt dabei jeweils *Events* in ADT, die an den Plug&Play-Dienst weitergeleitet werden. Dieser kann anhand der Events nun entsprechende Agenten starten oder aktualisieren, und ihnen ebenfalls Informationen über ihre Nachbarschaft bzw. Kommunikationspartner mitteilen, da diese ebenfalls aus dem digitalen Zwilling in ADT hervorgehen.

Für den Rückkanal, d. h. Stellsignale an die Aktorik, werden Cloud-to-Device-Nachrichten über den

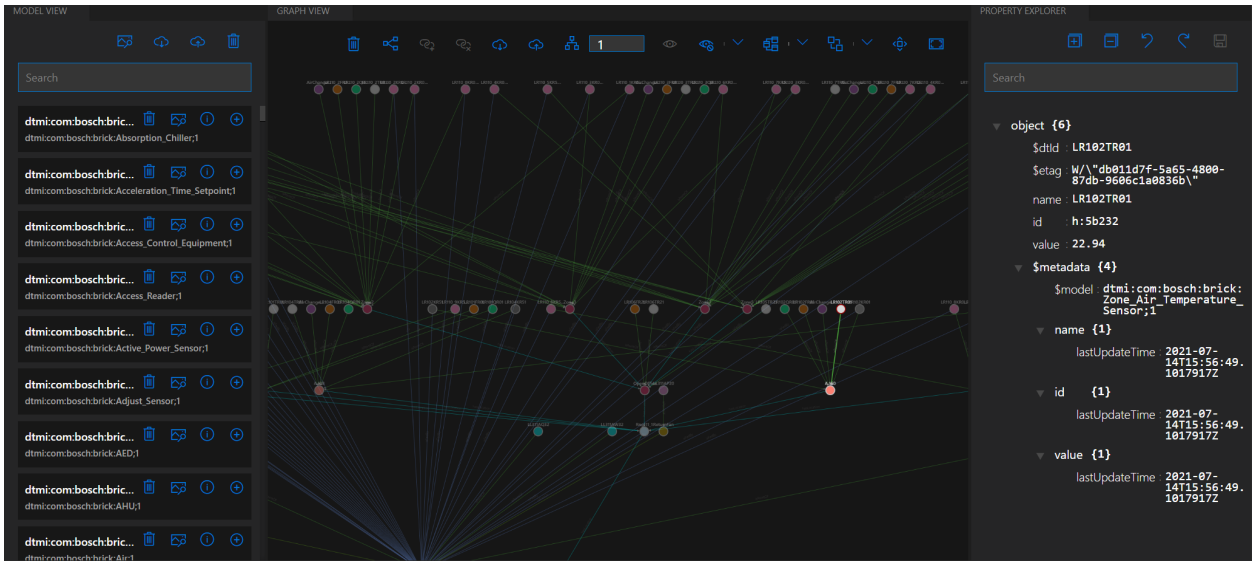


Abbildung 6.47: Darstellung des digitalen Zwillings von Gebäude Rng111 (Ausschnitt)

IoT-Hub an die Steuerung im Gebäude gesendet, die diese Nachrichten entsprechend verarbeitet.

6.8.3 Azure-Anbindung

Um externen Zugriff auf die Systeme der Gebäudeautomation zu erhalten, wurde das Gebäude an die Azure-Cloud angebunden. Der Informationsfluss für einen beispielhaften Agenten zur Zonenregelung von Raum A.160 ist in Abbildung 6.48 dargestellt.

Die verschiedenen Datenpunkte bzw. Messsignale werden von dem Jace-Controller als Telemetrie an den IoT Hub in Azure verschickt. Dort werden sie im Rahmen einer Azure Function verarbeitet und die jeweiligen Twins gemäß Unterabschnitt 6.8.2 erstellt bzw. aktualisiert. Dies erzeugt ein Event innerhalb von ADT, das den Versand der neuen Information an den entsprechenden Agenten auslöst. Wenn zum Beispiel ein neuer Wert für die Raumtemperatur von Raum A.160 eintrifft, wird der gesamte Teilgraph mit allen Sensoren/Aktoren des Raums A.160 an den entsprechenden Raumagenten versandt, der sich ebenfalls mit dem IoT Hub über den in Unterabschnitt 6.3.3 beschriebenen Konnektor verbunden hat. Der Agent berechnet einen neuen Stellwert und schickt ihn über den IoT Hub an die Gebäudesteuerung.

Alternativ kann der Raumagent auch zyklisch arbeiten und am Beginn jeden Zyklus die relevanten Informationen aus dem Graphen aktiv abfragen. Damit entfällt die Notwendigkeit, die Events in ADT zu verarbeiten und das System arbeitet nicht mehr ereignisgesteuert, sondern mit festen Zeitschritten.

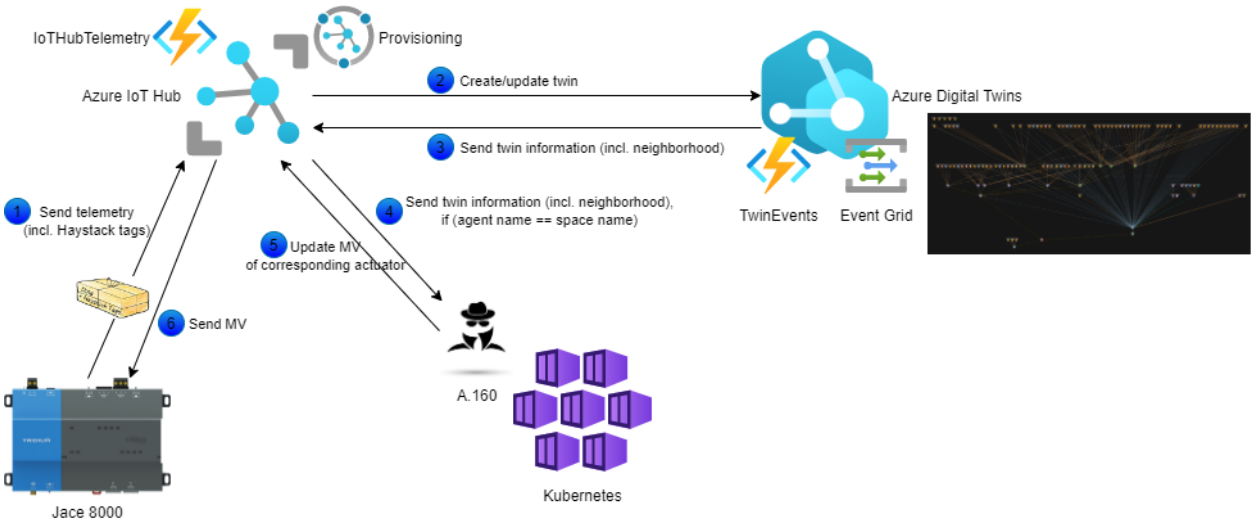


Abbildung 6.48: Schematische Darstellung des Informationsflusses

6.8.4 Plug'n'Play Service und CloneMap

Die Funktion des Plug&Play-Dienstes ist die Erstellung von Agenten für definierte Entitäten, wie zum Beispiel Räume. Dazu wird der in Unterabschnitt 6.8.2 erzeugete digitale Zwilling des Gebäudes in *Azure Digital Twins* (ADT) genutzt, um automatisch die Konfiguration der Agenten zu generieren.

Der generelle Aufbau des Plug&Play-Dienstes ist in Abbildung 6.49 dargestellt. In der realen Anwendung kann ein Systemadministrator mit nötigen Berechtigungen über eine REST-API HTTP Anfragen an den Plug&Play-Dienst stellen. Der Plug&Play-Dienst ist über das Framework *FastAPI* in Python implementiert. Die Endpunkte der Schnittstelle für Anwender können in der automatisch erzeugten Swagger-Dokumentations des Dienstes nachgesehen werden.

Im Folgenden werden die einzelnen Optionen für Anfragen an den Dienst erläutert:
Der Endpunkt *start_mas* dient zum Starten des Multi-Agenten-Systems in *cloneMAP*. Hierzu ist der Name, Hostname, sowie Image-Name und Pull-Secret zum Herunterladen des Docker-Images anzugegeben. Mit diesen Informationen wird ein neues MAS durch *cloneMAP* gestartet, allerdings noch ohne Agenten.

Über ein POST-Befehl an den Endpunkt *create_agents* kann eine Liste an Raum-Namen angegeben werden, für die Agenten konfiguriert und gestartet werden sollen. Intern wird im Plug&Play-Dienst anschließend die vorliegende Semantik genutzt, um auf Basis der Raum-Namen alle Modul-Konfigurationen zu erstellen. An dieser Stelle sei darauf verwiesen, dass für einen anderen Anwendungsfall diese Logik neu implementiert werden muss. Hierbei sei noch auf das Folgeprojekt AGENT-2 verwiesen, bei dem die Modellierung der Räume ebenfalls automatisiert werden soll. Sobald die Logik aber einmal vorhanden ist, skaliert der Dienst direkt. Somit ist das Aufsetzen von einem Raum-Agenten ebenso aufwendig wie das Aufsetzen von neun Agenten, und erfordert lediglich eine Anfrage an den Dienst. Der einzige Unterschied ist die Payload, welcher an den Endpunkt

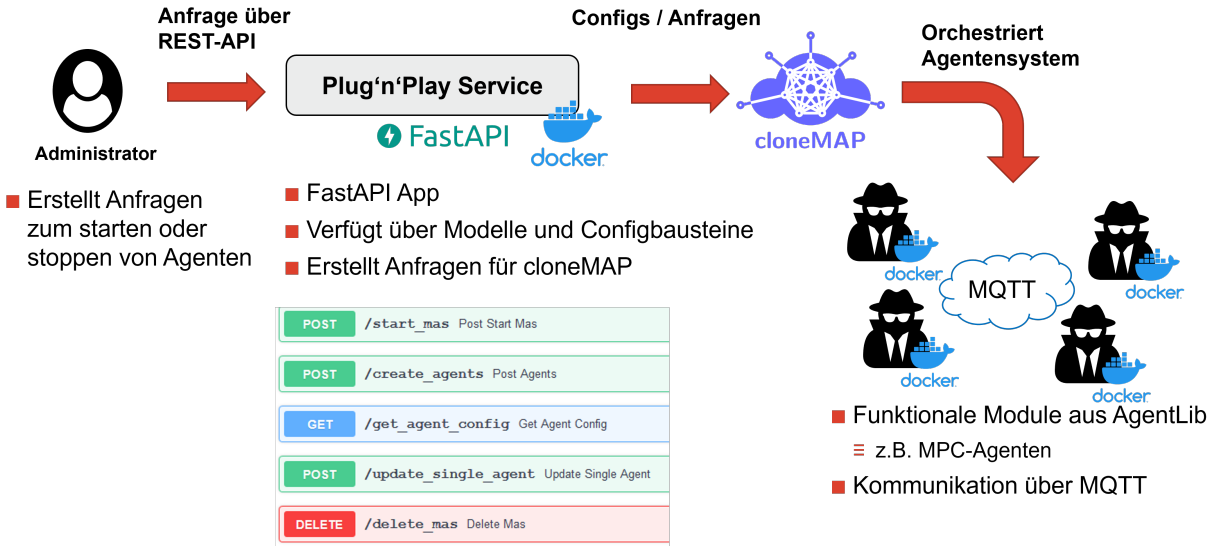


Abbildung 6.49: Schematischer Ablauf einer Anfrage über den Plug'n'Play-Dienst.

create_agents geschickt wird. Dieser Unterschied wird durch die beiden Folgenden Auflistungen visualisiert:

Auflistung 6.11: HTTP-Payload für die Regelung von einem Raum

```
{
    "rooms": ["A.108"],
    ...
}
```

Auflistung 6.12: HTTP-Payload für die Regelung von neun Räumen

```
{
    "rooms": [
        "A.108", "A.149", "A.155", "A.160",
        "A.162", "Zone2", "Zone3", "Zone5", "Zone7"
    ],
    ...
}
```

Weitere optionale Einstellungen sind Kostenfunktionen, Sicherheitsfeatures und Koordinator-Einstellungen. Über HTTP-Anfragen an *cloneMAP* werden die Agenten entsprechend auf verteilten virtuellen Docker-Instanzen gestartet.
Fällt dem Administrator im Betrieb auf, dass die Agenten nicht wie gewünscht agieren, können die nachfolgenden Optionen des Plug&Play-Dienstes genutzt werden: Über den Endpunkt *get_agent_config* kann für einen bestimmten Raum die erstellte Agentenkonfiguration aus *cloneMAP* abgefragt werden. Sollten Einstellungen, z.B. der Sollwert der Raumtemperatur, nicht stimmig sein, kann dieser einzelne Agent über den Endpunkt *update_single_agent* im Betrieb geändert

werden.
Sollte das Agentensystem ausgeschaltet werden, kann der Endpunkt *delete_mas* genutzt werden.

6.8.5 Ergebnisse

An dem Demonstrator wurden Versuche sowohl im Sommer als auch im Winter durchgeführt. In Abbildung 6.50 ist beispielhaft der Verlauf des Raums A.160 zu sehen, während eines Versuchs, der im Sommer 2022 durchgeführt wurde. Dabei wurden jeweils ein Agent für die neun thermischen Zonen und ein Agent für eine virtuelle Lüftungsanlage erstellt. Die Zonen-Agenten besitzen jeweils ein physikalisches Modell, welches die Entwicklung der Zonentemperatur und des CO_2-Gehalts voraussagt, abhängig des Zuluftvolumenstroms. Der Agent für die Lüftungsanlage hingegen kennt die Kosten, die durch die Bereitstellung des jeweiligen Luftmassenstroms verursacht werden. Da die CO_2-Messung in der Abluft stattfindet, ist ein gewisser Mindestluftvolumenstrom notwendig, und die Messung wird genauer bei großen Luftvolumenströmen. Zum Zeitpunkt der Versuche war die Belegung der Büros in Renningen sehr gering, da ein Großteil der Mitarbeiter auch während dem Abklingen der Pandemie häufig im Home-Office war. In Abbildung 6.51 ist auf der linken Seite der Temperaturverlauf in einigen der untersuchten Zonen zu sehen, und auf der rechten Seite der Verlauf der CO_2-Konzentrationen.

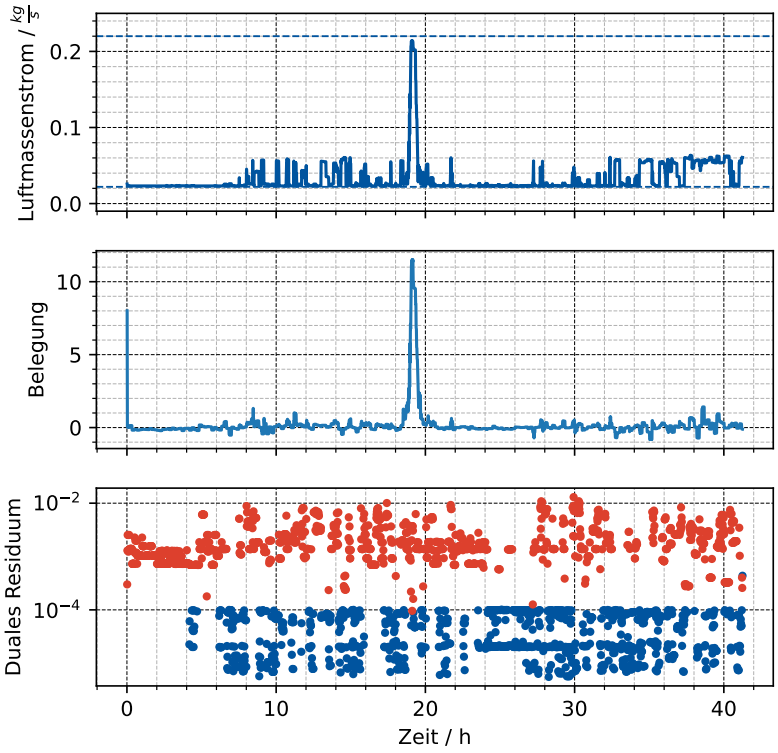


Abbildung 6.50: Ergebnisse der verteilten MPC für Raum A.160 im Sommer. Oben: Stellwert für die Luftklappe in $\frac{kg}{s}$. Mitte: Geschätzte Belegungszahl Unten: Duales Residuum des ADMM-Algorithmus.

Ebenfalls zeigt die Abbildung, dass die Sensorik mit spürbarem Messrauschen arbeiten muss, welches von Sensor zu Sensor verschieden ist. Vor Beginn der Versuche lief die Lüftung mit ca. 19 °C Zulufttemperatur bei dreifachem Luftwechsel, sodass die Innenraumtemperatur trotz Außentemperaturen über 30 °C deutlich unter der definierten Grenze für den thermischen Komfort (25°C lag). Zum Zeitpunkt der Versuche stellten damit sowohl der thermische Komfort, als auch die Luftqualität keine großen Anforderungen an die Lüfterleistung. Somit ist der optimale Betrieb des Raums bei minimalem Luftvolumenstrom gegeben, um Energie einzusparen. Auf dem oberen Graph ist der tatsächliche Zuluftvolumenstrom zu sehen, sowie der minimale Zuluftstrom in gestrichelt. Es ist zu sehen, dass der Zuluftvolumenstrom, der durch die verteilte MPC eingeregelt wird, den größen Teil der Zeit nahe am minimalen Luftvolumenstrom verharrt. Eine deutliche Spitze im Zuluftvolumenstrom ist bei ca. 19 h Versuchsdauer zu erkennen. Zu diesem Zeitpunkt wurde im Besprechungsraum A.160 eine künstliche Erhöhung der CO_2-Konzentration verursacht, indem mit einer handelsüblichen CO_2-Kartusche CO_2 in den Raum eingelassen wurde. Der Regler reagiert auf diese Störung und erhöht zügig den Luftmassenstrom auf den Maximalwert. Als Resultat ist es möglich, trotz der schnellen Störung die CO_2-Konzentration unterhalb 900 ppm zu halten, wie in Abbildung 6.51

rechts im Bereich von 10 Uhr zu sehen ist.

Auf dem mittleren Graphen von Abbildung 6.50 ist die geschätzte Belegungszahl des Raums A.160 zu sehen, die dynamisch aus den Messwerten für die CO_2-Konzentration von Zu- und Abluft, sowie des Luftvolumenstroms berechnet wird. Die aufgrund des Messrauschens ist die Belegung fast, aber nicht ganz null. Während der CO_2-Einspritzung wird die Belegung auf bis zu 11 Personen geschätzt.

Auf dem unteren Graphen von Abbildung 6.50 ist das duale Residuum des ADMM-Algorithmus abgebildet. Dabei sind rot Punkte, bei denen das Iterationslimit von 40 erreicht wurde und die Optimierung so abgebrochen, während in blau Punkte sind, bei denen die notwendige Toleranz von 10^{-4} erreicht wurde, und der Algorithmus bereits nach weniger Iterationen terminiert werden konnte. Es ist auffällig, dass die Konvergenz des Algorithmus stochastisch weit verteilt ist, und Punkte mit guter und schlechter Konvergenz direkt nebeneinander zu finden sind. Nach ca. 25 h findet sich ein Bereich, in dem die meisten Optimierungen konvergieren. Dies deckt sich mit einem Bereich, in dem auch die Personenschätzung kaum schwankt. Es ist ersichtlich, dass bei einer verteilten MPC mit ADMM noch zusätzliche Arbeit in die Stabilität investiert werden muss, wenn mit Realdaten gearbeitet wird. Dabei können Filtermethoden für die Eingangsdaten, oder generelle Verbesserungen am Algorithmus in Frage kommen. In Abbildung 6.52 sind noch Ergebnisse, aus

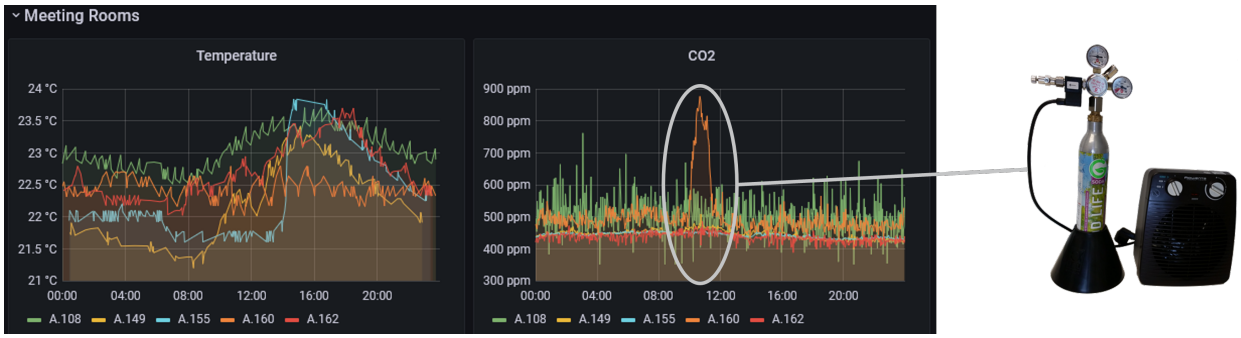


Abbildung 6.51: Monitoringdaten für Temperatur und CO_2-Konzentration während der Regelung mit verteilter MPC.

einem Versuch während der Heizperiode zu sehen. Auch hier war die Belegung der Räume klein, so dass keine große Lüfterleistung nötig war um die Luftqualität zu gewährleisten. Entsprechend wird auch hier der minimale Volumenstrom gewählt. Ein Blick auf die geschätzte Belegungszahl (untere Grafik) zeigt, dass die Personenschätzung stark schwankt bei geringem Luftmassenstrom. Verglichen mit dem Versuch im Sommer, wurde hier für die Schätzung die Anpassung vorgenommen, dass keine negative Belegungszahl berechnet werden kann.

6.9 Zusammenfassung und wichtigste Erkenntnisse

Im Rahmen des Projekts wurde eine prototypisches agentenbasiertes Gebäudeenergiemanagementsystem entwickelt und an einem Demonstrator getestet. Dabei konnten verschiedene Softwarebausteine kombiniert werden, um das finale Agentensystem am realen Gebäude auszuführen. Konkret

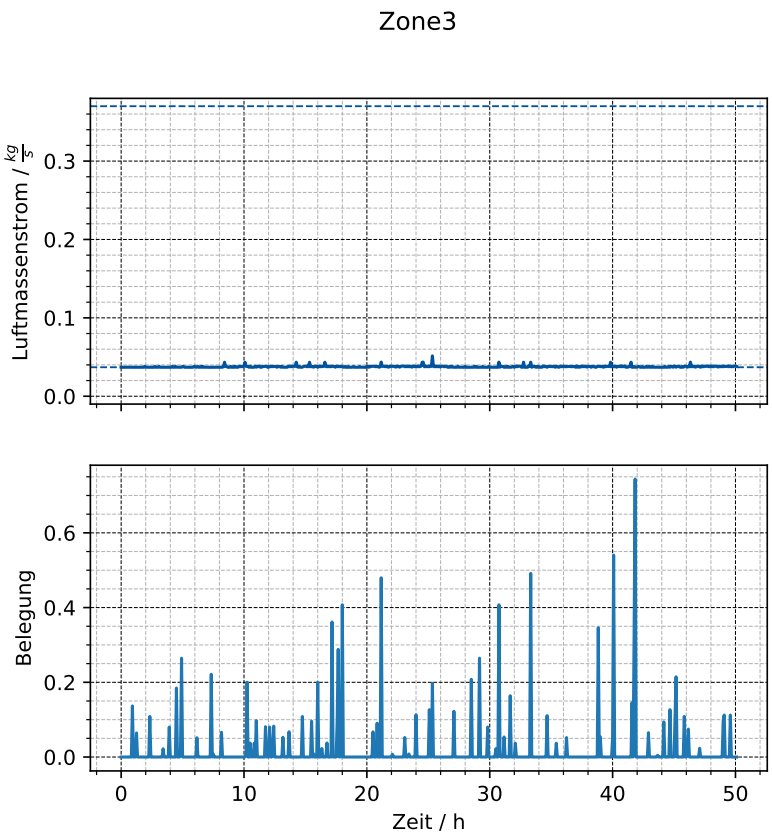


Abbildung 6.52: Ergebnisse für die Regelung im Winter. Aufgrund der geringen Belegung der Büros ist eine Lüftung nicht notwendig, und der Luftmassenstrom wird auf ein Minimum gefahren. Dies beeinflusst die CO_2 basierte Personenschätzung negativ.

wurde eine eigene Bibliothek zur Strukturierung und Kommunikation der Agenten entwickelt - die *AgentLib*. Die Agenten der *AgentLib* wurden über *cloneMAP* auf Docker-Containern ausgeführt. Für die Anbindung an das reale Gebäude wurde eine Kommunikation über eine Azure-Cloud in der *AgentLib* implementiert. In der Azure-Cloud wiederum wurden Funktionen hinterlegt, die über eine SPS auf das BACNet des Demonstratorgebäudes lesen und schreiben konnten. Schließlich wurde ein Web-Service basierend auf FastAPI geschrieben, mit dem sich das Energiemanagement für die Demonstratoretage für eine beliebige Anzahl an Räumen einfach per HTTP-Request starten ließ.

Mithilfe der *AgentLib* konnte die Komplexität der Untersuchungen langsam gesteigert werden. So erlaubte ihre modulare Struktur, dass funktionale Agenten an einfachen simulativen Beispielen entwickelt und getestet werden konnten, und dann durch Austausch einzelner Komponenten in die oben erwähnte Realanwendung integriert werden konnten.

Letztlich gelang die prototypische Anwendung von verteilten modellprädiktiven Regelungskonzepten, sowohl simulativ als auch experimentell, und mit verschiedenen Algorithmen. Im Zuge der Realanwendungen wurden neue Herausforderungen für den Durchbruch der Methode in die Pra-

xis erkannt. So ist insbesondere die Modellierung der Energiesysteme entscheidend für den Erfolg von prädiktiven Regelungsverfahren. Ebenso bereitete die Ungenauigkeit von Sensoren und Aktoren Schwierigkeiten, die in der Simulation nicht auftraten. Für die Lösung dieser Probleme wird große Hoffnung in das Nachfolgeprojekt gelegt, bei dem mit Methoden des maschinellen Lernens Realdaten für die Prozessführung genutzt werden sollen.

7 Notwendigkeit und Angemessenheit der geleisteten Arbeit

In diesem Kapitel werden die Arbeiten der verschiedenen Partner kurz dargestellt. Die jeweiligen Beschreibungen beziehen sich auf die folgenden Arbeitspakete des Vorhabens.

AP1: Vorarbeiten, grundsätzliche Systemarchitektur sowie technische Grundlagen und Anforderungen

AP2: Entwicklung der Modellbibliotheken

AP3: Entwicklung eines Software-in-the-Loop-Frameworks

AP4: Entwicklung der Agenten

AP5: Begleitende Simulationen und Experimente

AP6: Nutzermodelle und adaptive Komfortagenten

AP7: Systemarchitektur und Prototypentwicklung

AP8: Implementierung/Demonstration in einem realen Gebäude inkl. Monitoring

AP9: Systematische Analyse, Bewertungskriterien und Vergleich mit vorhandenen Lösungen

AP10: Verbreitung und Verwertung der Ergebnisse sowie Projektmanagement

7.1 RWTH

Dieser Abschnitt beschreibt die wichtigsten von der RWTH Aachen in den verschiedenen Arbeitspaketen geleisteten Arbeiten und stellt sie in Bezug zu der Notwendigkeit und Angemessenheit der durchgeführten Aktivitäten.

AP1: Im ersten Arbeitspaket wurde eine Literaturrecherche durchgeführt, sowie der Wissensstand der Partner untereinander abgeglichen. Darauf aufbauend wurde ein gemeinsames Verständnis über die Aufgaben und Funktionen eines Multi-Agenten-Systems entwickelt.

AP2: Für die Entwicklung der Agentenmethode und der Regelungsalgorithmen, sowie die Anwendung am Demonstrator, waren fortlaufende Tests und Simulationen notwendig. Dabei muss zwischen Modellen unterschieden werden, die für die Optimierung geeignet sind, und solchen, die detailliert realitätsnahe Prozesse abbilden. Im Rahmen des Arbeitspaket 2 wurden diese entwickelt.

AP3: Im Arbeitspaket drei sollte ein Software-in-the-Loop Framework entwickelt werden. Dieses wird benötigt, um die Agentenmethode simulativ zu testen und entwickeln, bevor sie an einem echten Gebäude angewandt wird. Im Projektantrag wurden dafür die Entwicklungsumgebung JADE als JAVA-basiertes Framework vorgeschlagen. Da die Forschungsgemeinschaft jedoch deutlich aktiver in Python ist und insbesondere moderne quelloffene Werkzeuge zur für Modellierung von Energiesystemen, numerische Optimierung und Regelung gut über Python eingebunden werden können, wurde es als notwendig erachtet, ein Python-basiertes Software-in-the-Loop Framework zu nutzen. Dieses Framework - genannt *AgentLib* - wurde im Rahmen von Arbeitspaket 3 entwickelt. Dabei wurde die Einbindung des Functional-Mockup-Interfaces, ein sich zunehmend verbreitender Standard, der auch von Modelica unterstützt wird, vorgenommen. Durch eine modulare Struktur kann dasselbe Framework für Simulationen und Experimente genutzt werden. Das entwickelte Framework wird im Folgeprojekt genutzt werden und wird auch bereits für andere Forschungsvorhaben genutzt.

AP4: Im vierten Arbeitspaket sollten Agenten entwickelt werden, welche verschiedene Regelungen und Funktionalitäten beinhalten können. Die Agenten sind somit in Kombination mit der AgentLib ein Kern-Element des Projektes. Die Agenten wurden dabei als sogenannte Module in die *AgentLib* eingebunden. Es wurde eine Auswahl an Agenten entwickelt, unter anderem zur klassischen Regeleung mittels PID- und Hysteresereglern, sowie modellprädiktive Regler und verteilte Regler. Ebenso wurden einige Zusatzmodule entwickelt, die Hilfsfunktionen umfassen. Dazu zählen vor allem der Simulator, der FMU-Modelle und handgeschriebene Modelle ausführen kann, sowie Module für die Anbindung an IoT-Plattformen wie Fiware, zur Komfortbestimmung oder zur Störgrößenschätzung. Die Agenten kommunizieren über eine einheitliche Datenstruktur. Mit der entwickelten Auswahl an Agenten konnten die simulativen und experimentellen Versuche sowie die Demonstration am einem Gebäude durchgeführt werden.

AP5: Im fünften Arbeitspaket wurden begleitende Simulationen und Experimente durchgeführt. Diese waren notwendig für die Entwicklung der Software in Arbeitspaket 3 und 4, sowie zur Bewertung der Regelungsmethoden. So konnten über kleinere Beispiele die Methoden getestet und weiterentwickelt werden, um auf die Demonstration hinzuarbeiten. Außerdem mussten Erweiterungen am Prüfstand SenkenHiL der RWTH getätigt werden. An dem Prüfstand konnte die Ausführung eines Agentensystems - angebunden an realer Hardware - getestet werden. Die daraus gewonnen Erfahrungen flossen in die Demonstration am Realgebäude ein.

AP6: Im sechsten Arbeitspaket wurde die Einbindung von Komfortagenten in das Agentensystem untersucht. Hierbei wurden verschiedene Komfortmodelle, beispielsweise basierend auf dem PMV, oder detaillierten Körpersimulationen, implementiert. Neben der Regelung der Anlagentechnik ist die Berücksichtigung des Nutzerkomforts essenziell für den Betrieb von Gebäuden und ebenso für die Akzeptanz moderner Automationssysteme. Die Entwicklung von Komfortagenten ist somit ein essenzieller Bestandteil des gesamten Agentensystems. Insbesondere die Flexibilität und Modularität der *AgentLib*, verschiedene funktionale Module

neben der Regelung zu unterstützen, wurde im Rahmen dieses Arbeitspakets gezeigt.

AP7: Im siebten Arbeitspaket, wurde die verteilte Ausführung des Agentensystems über *cloneMAP* realisiert. Da *cloneMAP* die Ausführung in Containern sowohl lokal als auch über Kubernetes ermöglicht, ist somit die Ausführung auf modernen Cloud-Systemen demonstriert worden. Die Überführung der Agenten von lokaler Simulation auf eine Plattform, die für eine Realanwendung geeignet ist, ist ein entscheidender Schritt bei der Demonstration der Praxistauglichkeit der agentenbasierten Regelung. Hierbei ist insbesondere hervorzuheben, dass die Agenten erst lokal entwickelt und Fehler behoben wurden und dann über *CloneMAP* auf verteilten Recheninstanzen ausgeführt wurden. Mithilfe eines Plug'n'Play-Services konnte die Erstellung und Ausführung des Agentensystems für den Demonstrator in Renningen noch vereinfacht werden.

AP8: Im achten Arbeitspaket wurde die Demonstration an einem Gebäude in Renningen durchgeführt. Die reale Anwendung ist zwingend notwendig, um die Funktionalität der aufgebauten Infrastruktur zu validieren und die Anforderungen an komplexe Gebäudesysteme mit zu berücksichtigen. Die RWTH hat hierbei die Ausführung von Agenten mithilfe der *AgentLib* und *cloneMAP* begleitet.

AP9: Im Rahmen dieses Arbeitspakets wurden die Ergebnisse der Simulationen und Regler ausgewertet, visualisiert und eingeordnet. Die kontinuierliche Auswertung ist nötig für fortlaufende Entwicklung und Verbesserung der Agentenmethode.

AP10: Das zehnte Arbeitspaket umfasste Veröffentlichungen und Organisation. Hier wurde die Koordination der Projektpartner und eine gute Zusammenarbeit sichergestellt, sowie die Ergebnisse des Projekts auf internationalen Konferenzen vorgestellt. Dabei wurden von der RWTH zwei Publikationen in Fachzeitschriften veröffentlicht. Außerdem wurden Beiträge zu sechs internationalen Konferenzen und einer nationalen Konferenz getätigt, die ebenfalls als Publikation in den Proceedings einzusehen sind.

7.2 FAU

Die Arbeiten der FAU lassen sich wie folgt darstellen:

AP1: Es wurden die Anforderungen von verteilter MPC an die Systemarchitektur untersucht. Weitergehend wurden erste Algorithmen getestet, die sich zur verteilten MPC eignen.

AP2: Hier wurden die Projektpartner bei Bedarf unterstützt.

AP3: Die Kommunikationsanforderungen für verteilte MPC wurde miteingebracht.

AP4: Die infrage kommenden Algorithmen für verteilte MPC wurden hier untersucht. Hieraus hat sich ADMM als derzeit robustestes Verfahren herauskristallisiert welches somit in der Agentlib von der RWTH Aachen implementiert wurde. Zusätzlich wurde der SENSI Ansatz weiterentwickelt, da dieser vielversprechende Eigenschaften für die verteilte MPC besitzt.

AP5: Der SENSI Algorithmus aus AP4 wurde in der Agentlib implementiert, um einen Vergleich mit den anderen untersuchten Algorithmen zu ermöglichen. Hier hat sich auch die Notwendigkeit eines festen Koordinatoragenten für die Entwicklung und Erprobung gezeigt, bei der konzeptionell mitgearbeitet wurde.

AP6: Hierzu wurden keine Arbeiten angefertigt, da der Fokus auf der Entwicklung von verteilten MPC Ansätzen gesetzt wurde.

AP7: Wie in AP6 wurden hierzu keine Arbeiten angefertigt.

AP8: Vereinfachte Grey-Box Modelle für die MPC wurden erstellt, die die Dynamik der untersuchten Demonstratoren entspricht.

AP9: Hier wurde maßgeblich der Rechen- und Kommunikationsbedarf der Algorithmen untersucht und verglichen. Zusätzlich wurde bei der Interpretierung und Visualisierung der Ergebnisse unterstützt.

AP10: Die Ergebnisse des SENSI Algorithmuses wurde auf Konferenzen vorgestellt.

7.3 Bosch

Die Schwerpunkte seitens Bosch in den jeweiligen Arbeitspaketen sind im Folgenden dargestellt.

AP1: Die Robert Bosch GmbH hat im Rahmen des Projekts ein Gebäude am Forschungscampus Renningen als Demonstrationsobjekt zur Verfügung gestellt. Für die weiteren Arbeitspakete wurden entsprechende bauphysikalischen und betrieblichen Daten recherchiert, gesammelt und bereitgestellt. Zusätzlich wurde ein BIM-Modells des Demonstratorgebäudes zur Verfügung gestellt.

Innerhalb des Konsortiums wurden mögliche Architekturen von Agentensystemen inklusive der Anbindung an reale Systeme diskutiert und letztlich die Grundlage für die folgenden Arbeitspakete gelegt.

Weiterhin wurde eine Literaturrecherche für verteilte optimierungsbasierte Regelungen durchgeführt. Darauf aufbauend wurde der wissenschaftliche Stand der verteilten modellprädiktiven Regelung aufbereitet und im Konsortium präsentiert. Dies half dabei, für die betrachtete Anwendung geeignete Algorithmen auszuwählen (Unterabschnitt 6.1.5).

AP2: Im Rahmen dieses Arbeitspakets wurde ein Modell der betrachteten Zonen des Demonstratorgebäudes in MATLAB/Simulink entwickelt, parametriert und plausibilisiert. Dieses Modell bildet die Grundlage für die jeweiligen Reglermodelle innerhalb der Agenten im Rahmen der modellbasierten Regelung.

AP3: Für die Umsetzung der dezentralen Topologie aus Softwareagenten wurde in diesem Arbeitspaket die *AgentLib* entwickelt. Dabei wurde konzeptionell bei Designentscheidungen mitgearbeitet und die spätere Implementierung unterstützt. Insbesondere wurde die Anbindung an

die Azure-Cloud implementiert, um darüber bidirektionalen Zugriff auf das reale Gebäude zu erhalten.

Außerdem wurde ein einfaches thermisches Knotenmodell entwickelt, welches als Grundlage für die Regelungsagenten benutzt wurde.

AP4: Das Konsortium wurde bei der Entwicklung einer möglichen Agentenstruktur, der Implementierung erster Proof-of-Concept-Agenten und einer Entwicklungs- und Testumgebung zur lokalen Ausführung von Agenten unterstützt.

Aufbauend auf dem in AP1 erarbeiteten wissenschaftlichen Stand der verteilten modellprädiktiven Regelung wurden verschiedene Regelungsansätze diskutiert und ausgewählt. Daraufhin wurde ein verhandlungsbasierter verteilter modellprädiktiven Regelungsansatz (Unterabschnitt 6.6.4) sowie eine hierarchische verteilte modellprädiktive Regelung entwickelt und implementiert (Unterabschnitt 6.6.1). Zusätzlich wurde bei der Erstellung von Koordinatoren für verteilte Algorithmen mitgearbeitet.

Weiterhin wurde das Konsortium bei der Implementierung eines Minimalbeispiels für den Test der Regelungsansätze unterstützt, indem ein thermisches 4-Raum-Modell in *PyOdeSys* und *casADi* implementiert wurde und die im Python-Paket *disropt* verfügbaren Algorithmen evaluiert wurden.

AP5: Die *AgentLib* wurde zur Modellierung von Systemen am Forschungscampus Renningen genutzt, um die Anwendbarkeit bzw. Nutzbarkeit der Bibliothek zu validieren. Hierfür wurden bewusst Kollegen eingebunden, welche nicht an der Entwicklung der AgentLib beteiligt waren, um eine unabhängige Einschätzung bezüglich Verbesserungspotentialen zu erhalten.

Zur Unterstützung der gemeinsamen Versuche am HIL-Prüfstand in Aachen wurde für diesen ein semantisches Modell erstellt (Unterabschnitt 6.7.2) und darauf aufbauend eine semiautomatische Erstellung der Konfiguration des Agentensystems realisiert. Die Versuche sowohl am Prüfstand, als auch in Renningen (Abschnitt 6.8), wurden gemeinsam geplant, unterstützt und die Ergebnisse ausgewertet.

AP6: Zu diesem AP haben keine Arbeiten stattgefunden.

AP7: Für die Prototypenentwicklung wurde ein Kubernetes-Cluster in Microsoft Azure erstellt und dort sowohl die FIWARE-Infrastruktur, als auch cloneMAP integriert.

Für die spätere Nutzung des Demonstrationsgebäudes wurden die Telemetriedaten der relevanten Zonen des Gebäudes über einen IoT Hub in die Azure-Cloud gesendet. Zur Ermöglichung einer automatischen Instanziierung der Agententopologie wurde eine semantische Beschreibung der Telemetriedaten basierend auf Haystack-Tags zu den Daten hinzugefügt. Durch ein Mapping zwischen Haystack- und Brick-Klassen und eine Konvertierung zwischen RDF und *Azure Digital Twins* konnte eine automatische Erstellung eines digitalen Zwillings des Gebäudes realisiert werden (Unterabschnitt 6.8.2).

Aufbauend darauf wurde am Entwurf und Implementierung eines *Plug&Play*-Dienstes zur automatischen Instanziierung von Agenten in cloneMAP basierend auf eingehender Telemetrie des Gebäudes mitgearbeitet (Unterabschnitt 6.8.4).

AP8: Aufbauend auf der im vorigen Arbeitspaket etablierten Cloud-Anbindung wurde zusätzlich ein Rückkanal implementiert, um aus der Cloud schreibend auf Stellgrößen zuzugreifen und so eine bidirektionale Anbindung hergestellt. Für die in Renningen durchgeführten Experimente am realen Gebäude wurde als Vorbereitung ein entsprechender IoT-Hub-Connector implementiert und in die *AgentLib* integriert.

AP9: Für die Bewertung der entwickelten Algorithmen wurde die hierarchische verteilte modellprädiktive Regelung am 4-Raum-Beispiel simulativ evaluiert und die Ergebnisse mit denen der anderen Algorithmen verglichen (Unterabschnitt 6.6.5).

AP10: Im Rahmen dieses Arbeitspaket wurden die administrativen Tätigkeiten zur Bearbeitung des Projekts durchgeführt. Eine Veröffentlichung zu der entwickelten Toolkette bezüglich des haystack-basierten Onboardings ist aktuell noch in Arbeit.

8 Nutzen und Verwertbarkeit der Ergebnisse

8.1 RWTH

Als zentrales Framework wurde im Rahmen des Projektes die *AgentLib* entwickelt, für die in Zukunft weitere Plugins geschrieben werden können, um zusätzliche Funktionalitäten einzubinden. Außerdem bietet das Plugin *AgentLib_MPC* bereits jetzt umfassende Funktionen für modellprädiktive Regelung, die auch im Folgeprojekt genutzt werden. In erster Linie wird dies im Nachfolgeprojekt AGENT-2 geschehen, das Framework wurde aber auch bereits in anderen Projekten wie DZWi (FKZ: 03EN1022B) genutzt. Ebenso bietet die *AgentLib* eine Grundlage für zahlreiche andere Forschungsvorhaben und Abschlussarbeiten am Lehrstuhl für Gebäude- und Raumklimatechnik. Für das Institut Automation of Complex Power Systems konnten weiterer Nutzen und Wartungsmöglichkeiten für die cloneMAP-Plattform gefunden werden. Die Veröffentlichung der *AgentLib* steht noch aus.

Neben der direkten Verwendung von entwickelter Software konnte ebenfalls Know-How im Bereich verteilter Rechensysteme, Netzwerkarchitektur und (verteilter) prädiktiver Regelung gesammelt werden. Dieses wird auch für zukünftige Projekte im Bereich der Gebäudeautomation wertvoll sein.

8.2 FAU

Der SENSI Algorithmus wird in einem weiterführenden DFG Projekt (Gr 3870/6-1) weiterentwickelt und in der Open-Source Toolbox GRAMPC-D implementiert. Zusätzlich wird dort die asynchrone Formulierung untersucht, um die Kommunikationszeiten weiter zu verringern.

8.3 Bosch

Auf dem Weg hin zu zukünftig proaktiven, autonomen Gebäuden stellt das *Onboarding*, d. h. die Erstellung eines digitalen Zwillings des Gebäudes, den momentan aufwändigsten, größtenteils manuellen Prozess dar. Die Art des digitalen Zwillings bzw. der benötigten Modelle umfasst dabei sowohl semantische Informationsmodelle, als auch dynamische Modelle zur Regelung der Gebäudeenergiesysteme.

Im Rahmen des Projekts wurde die in Unterabschnitt 6.8.2 dargestellte auf Haystack-Tags basierende Onboarding-Methode entwickelt, um ausgehend von Telemetriedaten des Gebäudes automatisch

ein semantisches Modell in *Azure Digital Twins* zu erstellen. In weiteren Schritten der entwickelten Toolkette wurde dieses semantische Model zur selbstständigen Parametrierung der Regleragenten verwendet (Unterabschnitt 6.8.4).

Es wird weiterhin untersucht, inwiefern diese prototypische Implementierung des automatischen Onboarding-Prozesses auch auf andere Projekte übertragen werden kann, um durch horizontale Skalierung eine produktiv einsetzbare Unterstützung dieses Prozesses zu erreichen.

Die aufwändige Erstellung von dynamischen Modellen ist bisher größtes Hindernis bei der Verbreitung höherer, modellbasierter Regelungsverfahren im Gebäudebereich. Bezüglich dieser Modelle wurden innerhalb dieses Projekts grundlegende physikalische Größen, wie zum Beispiel das Raumvolumen, aus einem semantischen Modell genutzt, um die entsprechenden Differentialgleichungen zu parametrieren. Dies ist ein erster Schritt hin zu einer einfacheren Anwendbarkeit der Algorithmen, jedoch für einen Durchbruch der Verfahren nicht ausreichend. Hier werden große Hoffnungen in das Nachfolgeprojekt AGENT-2 gesetzt, in dem datenbasiert Teilsystemmodelle verschiedener Komponenten erstellt werden.

9 Fortschritt auf dem Gebiet des Vorhabens bei anderen Stellen

Auf dem Gebiet der verteilen MPC mit Anwendung auf Gebäudeenergiesysteme, ist Lefebure et al. [2022] zu nennen. Das Projekt wurde vom schweizerischem Förderprogramm „Swiss Competence Centers for Energy Research“ unterstützt. Hier wurde eine verteilte MPC auf einen Demonstrator angewendet, die den Energieverbrauch verringert hat. Abgrenzend zu AGENT wurde dabei nicht die Konfiguration und Anwendbarkeit in generischen Gebäuden betrachtet.

Parallel zur Entwicklung der *AgentLib* wurde von Forschern der Universität Valencia die Plattform SPADE 3 entwickelt Palanca et al. [2020]. SPADE 3 ist als Open-Source-Paket verfügbar auf Github (`https://github.com/javipalanca/spade`). Die Veröffentlichung der *AgentLib* steht noch aus und soll während der Laufzeit des Folgeprojekts durchgeführt werden.

Die Ähnlichkeiten von SPADE 3 und der *AgentLib* bestehen darin, dass beide Pakete in Python implementiert sind, und es Nutzern ermöglichen, Funktionalitäten frei in ihren Agenten zu implementieren, und dabei das umfangreiche Ökosystem an anderen Python-Paketen zu nutzen. Die *AgentLib* hebt sich dabei in den folgenden Punkten von SPADE 3 ab:

- **Agenten-interne Modularität**: In SPADE 3 wird die Funktionalität eines Agenten direkt in eine Klasse implementiert, die von `Agent` erbt. In der *AgentLib* befindet sich der Entwicklungsaufwand in Modulen. Ein Agent wird aus mehreren Modulen zusammengesetzt, wodurch sein Verhalten einfach geändert werden kann.
- **Kommunikation**: SPADE 3 nutzt als Kommunikationsprotokoll XMPP, welches als etabliertes Nachrichtenprotokoll häufig für Textnachrichten verwendet wird, beispielsweise von WhatsApp. Der Payload einer Nachricht ist dabei XML-basiert. Die *AgentLib* ist nicht auf ein bestimmtes Kommunikationsprotokoll beschränkt. Standardmäßig unterstützt werden lokale Kommunikation (für Entwicklung und Debugging), sowie Python-Multiprocessing, sowie MQTT oder die Verbindung zu IoT-Plattformen wie Fiware. Der Payload ist dabei im `.json`-Format. Soll die Kommunikationsart geändert werden, muss lediglich das Kommunikatormodul ausgetauscht werden.
- **Konfiguration der Agenten**: Für Agenten in SPADE 3 erfolgt die Umsetzung eines Agenten vollständig in Python. Für *AgentLib*-Agenten erfolgt die Implementierung der Funktionen in Python, die Definition der Ein- und Ausgänge eines Agenten, sowie die Konfiguration der Module und Parameter eines Agenten erfolgt jedoch über eine `.json`-Konfigurationsdatei. Somit ist ein Agentensystem über serialisierbare, Mensch- und Maschinenlesbare Datei genau definiert.

Somit bietet die *AgentLib* insgesamt einen modulareren Aufbau, der die Entwicklung von komplexen Services über die initiale Phase bis zur realen Implementierung unterstützt, und sollte so vor allem für Forscher interessant sein. Im Gegenzug spricht für SPADE 3 die Umsetzung über asyncio, einer modernen Art und Weise, Python-Code für die parallele Ausführung von Tasks zu implementieren. Hier ist in Aussicht, dass der Kommunikations- und Ausführungsoverhead geringer ist, als bei der *AgentLib*, was eventuell bei der Ausführung von großen Multiagentensystemen zum Tragen kommen könnte.

10 Publikationen

Innerhalb des Projektzeitraums wurde eine Vielzahl an wissenschaftlichen Arbeiten veröffentlicht. Eine Übersicht ist in 10.1 zu finden.

Tabelle 10.1: Übersicht der wissenschaftlichen Veröffentlichungen im aktuellen Berichtzeitraum

Beitrag in:	**Titel:**	**Autoren:**
Proceedings of ECOS 2020 - The 33rd International Conference on Efficiency, Cost, Optimization, Simulation and Environmental impact of Energy Systems, June 29-July 3, 2020 Osaka, Japan	Auto-generation of hybrid automata for real-time operation optimization of building energy systems	Thomas Storek, Laurin Oberkirsch, Jonathan Kriwet, Marc Baranski, Thomas Schreiber and Dirk Müller
SoftwareX	OWL2Go: Auto-generation of Go data models for OWL ontologies with integrated serialization and deserialization functionality	Stefan Dähling, Lukas Razik, Antonello Monti
Autonomous Agents and Multi-Agent Systems	Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing	Stefan Dähling, Lukas Razik, Antonello Monti
Deutsche Kälte- und Klimatagung 2020	Virtueller Prüfstand zur Bewertung cloudbasierter Algorithmen für die Gebäudeautomation	Thomas Storek, Silas Koßler, Sazvan Saeed, Marc Baranski, Alexander Kümpel, Dirk Müller
Proceedings of CCTA 2021 - IEEE Conference on Control Technology and Applications, August 9-11, 2021, San Diego, USA	A sensitivity-based distributed model predictive control algorithm for nonlinear continuous-time systems	Hartwig Huber, Knut Graichen

17th International Conference of the International Building Performance Simulation Association (Building Simulation 2019): Bruges, Belgium, 1-3 September 2021	A virtual test bed for evaluating advanced building automation algorithms	Thomas Storek, Fabian Wüllhorst, Silas Koßler, Marc Baranski, Alexander Kümpel, Dirk Müller
Journal of Physics: Conference Series	Evaluation of linear and nonlinear system models in hierarchical model predictive control of HVAC systems	Steffen Eser, Phillip Stoffel, Alexander Kümpel, Dirk Müller
Journal of Physics: Conference Series	A tool for automated detection of hidden operation modes in building energy systems	Thomas Storek, Jonathan Kriwet, Alexander Kümpel, Dirk Müller
30th Mediterranean Conference on Control and Automation 2022	Distributed model predictive control of a nonlinear building energy system using consensus ADMM	Steffen Eser, Phillip Stoffel, Alexander Kümpel, Dirk Müller
IEEE Conference on Control Technology and Applications 2022	Comparison of Sensitivity-Based and ADMM-Based DMPC Applied to Building Automation	Hartwig Huber, Daniel Burk, Knut Graichen
Proceedings of the American Modelica Conference 2022, Dallas, Texas, USA, October 26-28	BESMod - A Modelica Library providing Building Energy System Modules	Fabian Wüllhorst, Laura Maier, David Jansen, Larissa Kühn, Dominik Hering, Dirk Müller

Literaturverzeichnis

Abdul Afram and Farrokh Janabi-Sharifi. Theory and applications of HVAC control systems – A review of model predictive control (MPC). *Building and Environment*, 72:343–355, February 2014. ISSN 03601323. doi: 10.1016/j.buildenv.2013.11.016.

H. Farooq Ahmad, Hiroki Suguri, Arshad Ali, Sarmad Malik, Muazzam Mugal, M. Omair Shafiq, Amina Tariq, and Amna Basharat. Scalable fault tolerant agent grooming environment: Sage. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, page 125–126, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930930. doi: 10.1145/1082473.1082816. URL https://doi.org/10.1145/1082473.1082816.

Muhammad Waseem Ahmad, Monjur Mourshed, Baris Yuce, and Yacine Rezgui. Computational intelligence techniques for HVAC systems: A review. *Building Simulation*, 9(4):359–398, August 2016. ISSN 1996-3599, 1996-8744. doi: 10.1007/s12273-016-0285-4.

Juan M. Alberola, Jose M. Such, Ana Garcia-Fornes, Agustin Espinosa, and Vicent Botti. A performance evaluation of three multiagent platforms. *Artificial Intelligence Review*, 34:145–176, June 2010. doi: 10.1007/s10462-010-9167-9.

Juan M. Alberola, Jose M. Such, Vicent Botti, Agustín Espinosa, and Ana García-Fornes. A scalable multiagent platform for large systems. *Computer Science and Information Systems*, 10:51–77, 2013. doi: 10.2298/CSIS111029039A. URL https://doi.org/10.2298/CSIS111029039A.

Joel A. E. Andersson, Joris Gillis, Greg Horn, James B. Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 2019.

Jupiter Bakakeu, Franziska Schäfer, Jochen Bauer, Markus Michl, and Jörg Franke. Building Cyber-Physical Systems - A Smart Building Use Case. In Houbing Song, Ravi Srinivasan, Tamim Sookoor, and Sabina Jeschke, editors, *Smart Cities*, pages 605–639. John Wiley & Sons, Inc., Hoboken, NJ, USA, June 2017. ISBN 978-1-119-22644-4 978-1-119-22639-0. doi: 10.1002/9781119226444.ch21.

Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*, volume 7. Wiley, 2007. ISBN 978-0-470-05747-6.

Anja Bestler and Knut Graichen. Distributed model predictive control for continuous-time nonlinear systems based on suboptimal admm. *Optimal Control Applications and Methods*, 40, June 2017. doi: 10.1002/oca.2459.

Anja Bestler and Knut Graichen. Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM: Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM. *Optimal Control Applications and Methods*, 40 (1):1–23, January 2019. ISSN 01432087. doi: 10.1002/oca.2459.

David Blum, Filip Jorissen, Sen Huang, Yan Chen, Javier Arroyo, Kyle Benne, Yanfei Li, Valentin Gavan, Lisa Rivalin, Lieve Helsen, Draguna Vrabie, Michael Wetter, and Marina Sofos. Prototyping The BOPTEST Framework For Simulation-Based Testing Of Advanced Control Strategies In Buildings. In *Building Simulation 2019*, pages 2737–2744, Rome, Italy, 2019. doi: 10.26868/25222708.2019.211276.

Stephen Boyd. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2010. ISSN 1935-8237, 1935-8245. doi: 10.1561/2200000016.

Stephen P. Boyd, Lin Xiao, Almir Mutapcic, and Jacob Mattingley. Notes on decomposition methods. In *Notes for EE364B, Stanford University*, 2007.

Daniel Burk, Andreas Völz, and Knut Graichen. A modular framework for distributed model predictive control of nonlinear continuous-time systems (grampc-d). 2020. doi: 10.48550/ARXIV. 2010.12315. URL https://arxiv.org/abs/2010.12315.

Jie Cai, Donghun Kim, Rita Jaramillo, James Braun, and Jianghai Hu. A general multi-agent control approach for building energy system optimization. *Energy and Buildings*, 127, May 2016. doi: 10.1016/j.enbuild.2016.05.040.

Clara Ceccolini and Roozbeh Sangi. Benchmarking Approaches for Assessing the Performance of Building Control Strategies: A Review. *Energies*, 15(4):1270, February 2022. ISSN 1996-1073. doi: 10.3390/en15041270.

Panagiotis D. Christofides, Riccardo Scattolini, David Muñoz de la Peña, and Jinfeng Liu. Distributed model predictive control: A tutorial review and future research directions. *Computers & Chemical Engineering*, 51:21 – 41, 2013.

Stefan Dähling, Lukas Razik, and Antonello Monti. Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing. *Autonomous Agents and Multi-Agent Systems*, 35 (1):10, April 2021. ISSN 1387-2532, 1573-7454. doi: 10.1007/s10458-020-09489-0.

A. Dorri, S. S. Kanhere, and R. Jurdak. Multi-Agent Systems: A Survey. *IEEE Access*, 6:28573–28593, 2018. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2831228.

Ján Drgoňa, Javier Arroyo, Iago Cupeiro Figueroa, David Blum, Krzysztof Arendt, Donghun Kim, Enric Perarnau Ollé, Juraj Oravec, Michael Wetter, Draguna L. Vrabie, and Lieve Helsen. All you need to know about model predictive control for buildings. *Annual Reviews in Control*, September 2020. ISSN 1367-5788. doi: 10.1016/j.arcontrol.2020.09.001.

Tobias Englert, Andreas Völz, Felix Mesmer, Sönke Rhein, and Knut Graichen. *A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)*, 2018. URL https://arxiv.org/abs/1805.01633.

H.J. Ferreau, C. Kirches, A. Potschka, H.G. Bock, and M. Diehl. qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.

Bernd Glück. Ein vorschlag zur verbesserten darstellung und messung der operativen raumtemperatur. *gi Gesundheitsingenieur - Haustechnik - Bauphysik - Umwelttechnik*, 127(2):76–82, 2006.

Alfonso González-Briones, Fernando De La Prieta, Mohd Saberi Mohamad, Sigeru Omatu, and Juan M. Corchado. Multi-Agent Systems Applications in Energy Optimization Problems: A State-of-the-Art Review. *Energies*, 11(8):1928, August 2018. doi: 10.3390/en11081928.

Sen Huang, Weimin Wang, Michael R. Brambley, Siddharth Goyal, and Wangda Zuo. An agent-based hardware-in-the-loop simulation framework for building controls. *Energy and Buildings*, 181:26–37, December 2018. ISSN 0378-7788. doi: 10.1016/j.enbuild.2018.09.038.

Hartwig Huber and Knut Graichen. A sensitivity-based distributed model predictive control algorithm for nonlinear continuous-time systems. In *2021 IEEE Conference on Control Technology and Applications (CCTA)*, pages 195–201, 2021. doi: 10.1109/CCTA48906.2021.9658733.

Hartwig Huber, Daniel Burk, and Knut Graichen. Comparison of sensitivity-based and admm-based dmpc applied to building automation. In *2022 IEEE Conference on Control Technology and Applications (CCTA)*, pages 546–553, 2022. doi: 10.1109/CCTA49430.2022.9966164.

Rawand E. Jalal and Bryan P. Rasmussen. Limited-communication distributed model predictive control for coupled and constrained subsystems. *IEEE Transactions on Control Systems Technology*, 25(5):1807–1815, 2017. doi: 10.1109/TCST.2016.2615088.

Jaewan Joe and Panagiota Karava. Agent-based system identification for control-oriented building models. *Journal of Building Performance Simulation*, 10(2):183–204, March 2017. ISSN 1940-1493, 1940-1507. doi: 10.1080/19401493.2016.1212272.

M. Killian and M. Kozek. Ten questions concerning model predictive control for energy efficient buildings. *Building and Environment*, 105:403–412, August 2016. ISSN 03601323. doi: 10.1016/j.buildenv.2016.05.034.

M. Killian and M. Kozek. Implementation of cooperative Fuzzy model predictive control for an energy-efficient office building. *Energy and Buildings*, 158:1404–1416, January 2018. ISSN 03787788. doi: 10.1016/j.enbuild.2017.11.021.

Kalliopi Kravari and Nick Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015a. ISSN 1460-7425. doi: 10.18564/jasss.2661. URL http://jasss.soc.surrey.ac.uk/18/1/11.html.

Kalliopi Kravari and Nick Bassiliades. A Survey of Agent Platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015b. ISSN 1460-7425. doi: 10.18564/jasss.2661.

Timilehin Labeodan, Kennedy Aduda, Gert Boxem, and Wim Zeiler. On the application of multi-agent systems in buildings for improved building operations, performance and smart grid interaction – A survey. *Renewable and Sustainable Energy Reviews*, 50:1405–1414, October 2015. ISSN 13640321. doi: 10.1016/j.rser.2015.05.081.

Nicolas Lefebure, Mohammad Khosravi, Mathias Hudobade Badyn, Felix Bünning, John Lygeros, Colin Jones, and Roy S. Smith. Distributed model predictive control of buildings and energy hubs. *Energy and Buildings*, 259:111806, 2022. ISSN 0378-7788. doi: https://doi.org/10.1016/j.enbuild.2021.111806. URL https://www.sciencedirect.com/science/article/pii/S0378778821010902.

Paulo Leitão, Stamatis Karnouskos, Luis Ribeiro, Jay Lee, Thomas Strasser, and Armando W. Colombo. Smart agents in industrial cyber–physical systems. *Proceedings of the IEEE*, 104(5): 1086–1101, 2016. doi: 10.1109/JPROC.2016.2521931.

J. M. Maestre and R. R. Negenborn. *Distributed Model Predictive Control Made Easy*. Springer, 2014.

J.M. Maestre, D. Muñoz de la Peña, E.F. Camacho, and T. Alamo. Distributed model predictive control based on agent negotiation. *Journal of Process Control*, 21(5):685–697, 2011. Special Issue on Hierarchical and Distributed Model Predictive Control.

Alexander Mentis and Levent Yilmaz. Validation and analysis of a distributed, agent-based metaheuristic for negotiation of consensus inspired by honeybee nest site selection behavior. pages 269–276, January 2014. doi: 10.1109/HICSS.2014.42.

Modelica Association Project "FMI" . Functional Mock-up Interface 2.0.2. Technical report, Modelica Association Project "FMI", December 2020.

Martin Möhlenkamp. *Thermischer Komfort bei Quellluftströmungen*. Dissertation, RWTH Aachen University, Aachen, 2019.

Maizura Mokhtar, Matt Stables, Xiongwei Liu, and Joe Howe. Intelligent multi-agent system for building heat distribution control with combined gas boilers and ground source heat pump. *Energy and Buildings*, 62:615–626, July 2013. doi: 10.1016/j.enbuild.2013.03.045.

Matthias A. Müller and Frank Allgöwer. Economic and distributed model predictive control: Recent developments in optimization-based control. *SICE Journal of Control, Measurement, and System Integration*, 10(2):39–52, 2017.

Javier Palanca, Andres Terrasa, Vicente Julian, and Carlos Carrascosa. SPADE 3: Supporting the New Generation of Multi-Agent Systems. *IEEE Access*, 8:182537–182549, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3027357.

Wei Peng and Qingfu Zhang. A decomposition-based multi-objective particle swarm optimization algorithm for continuous optimization problems. In *IEEE International Conference on Granular Computing*, pages 534–537, 2008. doi: 10.1109/GRC.2008.4664724.

Nicanor Quijano, Carlos Ocampo-Martinez, Julian Barreiro-Gomez, German Obando, Andres Pantoja, and Eduardo Mojica-Nava. The role of population games and evolutionary dynamics in distributed control systems: The advantages of evolutionary game theory. *IEEE Control Systems Magazine*, 37(1):70–97, 2017. doi: 10.1109/MCS.2016.2621479.

Kai Rewitz. *Modellierung des thermischen Komforts in Kabineninnenräumen*. Dissertation, RWTH Aachen University, Aachen, 2020.

Riccardo Scattolini. Architectures for distributed and hierarchical model predictive control – a review. *Journal of Process Control*, 19(5):723 – 731, 2009.

Holger Scheu and Wolfgang Marquardt. Sensitivity-based coordination in distributed model predictive control. *Journal of Process Control*, 21:715–728, 2011.

Mischa Schmidt and Christer Åhlund. Smart buildings as Cyber-Physical Systems: Data-driven predictive control strategies for energy efficiency. *Renewable and Sustainable Energy Reviews*, 90:742–756, July 2018. ISSN 13640321. doi: 10.1016/j.rser.2018.04.013.

Joanna Sendorek, Tomasz Szydlo, and Robert Brzoza-Woch. Software-Defined Virtual Testbed for IoT Systems. *Wireless Communications and Mobile Computing*, 2018:1–11, November 2018. ISSN 1530-8669, 1530-8677. doi: 10.1155/2018/1068261.

Gianluca Serale, Massimo Fiorentini, Alfonso Capozzoli, Daniele Bernardini, and Alberto Bemporad. Model Predictive Control (MPC) for Enhancing Building and HVAC System Energy Efficiency: Problem Formulation, Applications and Opportunities. *Energies*, 11(3):631, March 2018. ISSN 1996-1073. doi: 10.3390/en11030631.

Umar Siddiqui, Ghalib Ahmed Tahir, Attiq Ur Rehman, Zahra Ali, Raihan Ur Rasool, and Peter Bloodsworth. Elastic jade: Dynamically scalable multi agents using cloud resources. In *2012 Second International Conference on Cloud and Green Computing*, pages 167–172, 2012. doi: 10.1109/CGC.2012.60.

C.A. Silva, J.M.C. Sousa, T.A. Runkler, and J.M.G. Sá da Costa. Distributed supply chain management using ant colony optimization. *European Journal of Operational Research*, 199 (2):349–358, 2009. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2008.11.021. URL https://www.sciencedirect.com/science/article/pii/S0377221708010102.

SimPy-Team. Documentation for SimPy 4.0.2. https://simpy.readthedocs.io/en/latest/contents.html.

Rita Streblow. *Thermal Sensation and Comfort Model for Inhomogeneous Indoor Environments*. Dissertation, RWTH Aachen University, Aachen, 2010.

Rita Streblow and Dirk Müller. Noodel - 33-knoten-komfortmodell. *GI - Gebäudetechnik in Wissenschaft & Praxis*, 139(06):488–501, 2018.

Federico Tartarini and Stefano Schiavon. Pythermalcomfort: A Python package for thermal comfort research. *SoftwareX*, 12:100578, July 2020. ISSN 23527110. doi: 10.1016/j.softx.2020.100578.

Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1): 25–57, 2006. ISSN 0025-5610. doi: 10.1007/s10107-004-0559-y.

Michael Wetter. Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed. *Journal of Building Performance Simulation*, 4:185–203, September 2011. doi: 10.1080/19401493.2010.518631.

Fabian Wüllhorst, Thomas Storek, Philipp Mehrfeld, and Dirk Müller. Aixcalibuha: Automated calibration of building and hvac systems. *Journal of Open Source Software*, 7(72):3861, 2022. doi: 10.21105/joss.03861. URL https://doi.org/10.21105/joss.03861.

Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, June 1995. ISSN 0269-8889, 1469-8005. doi: 10.1017/S0269888900008122.

Fabian Wüllhorst, Laura Maier, David Jansen, Larissa Kühn, Dominik Hering, and Dirk Müller. BESMod - A Modelica Library providing Building Energy System Modules. In *American Modelica Conference 2022, Dallas, October 26-28*, pages 9–18, February 2023. doi: 10.3384/ECP211869.

Lei Xi, Zeyu Zhang, Linni Huang, and YU Tao. Wolf pack hunting strategy for automatic generation control of an islanding smart distribution network. *Energy Conversion and Management*, 122: 10–24, August 2016. doi: 10.1016/j.enconman.2016.05.039.

Bing Zhu, Lihua Xie, and Duo Han. Recent developments in control and optimization of swarm systems: A brief survey. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 19–24, June 2016. doi: 10.1109/ICCA.2016.7505246.