TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Maschinenwesen Institut für Energietechnik, Prof. f. Gebäudeenergietechnik u. Wärmeversorgung

Miguel Eduardo Manotas Ramos

Untersuchungen zur Simulation Raumlufttechnischer Anlagen zur Unterstützung von Planung und Betrieb von Nichtwohngebäuden

Investigation of simulation of Air Handling Units to support design and operation of non residential buildings

Dresden, 30.06.2021

TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Maschinenwesen Institut Energietechnik Professur für Gebäudeenergietechnik und Wärmeversorgung

Aufgabenstellung für die Projektarbeit im Forschungspraktikum Nr. 31/2020

Name des Studierenden: Miguel Manotas (Matr.-Nr. 4745300)
Studiengang: Maschinenbau
Studienrichtung: Energietechnik

Thema: **Untersuchungen zur Simulation Raumlufttechnischer Anlagen zur Unterstützung von Planung und Betrieb von Nichtwohngebäuden**
Investigation of simulation of Air Handling Units to support design and operation of non residential buildings

Aufgabenstellung:
Gebäude- und Anlagensimulation kann einen wesentlichen Beitrag zur Energieeffizienz von Gebäuden leisten, wenn sie bereits in zeitigen Planungsphasen angewendet wird. Eine wesentliche Hürde dabei ist der erhebliche Aufwand für die Erstellung der entsprechenden Simulationsmodelle. Durch die zunehmende Verbreitung von BIM (Building Information Modelling), d.h. die durchgängige Datenverarbeitung im Planungsprozess, kann dieser Modellierungsaufwand deutlich verringert werden. Für die Anlagensimulation ist die Anbindung an BIM ein Feld der aktuellen Forschung, wobei ein vielversprechender Lösungsansatz die Anwendung von Linked-Data- und Semantic-Web-Technologien ist.
Auf der anderen Seite besteht die Möglichkeit semantisches Wissen mit Hilfe von Machine Learning aus Messdaten zu generieren, Simulation können dafür synthetische Messdaten erzeugen.
Am Beispiel einer RLT-Zentralanlage im Neubau des Institutsgebäude des Fraunhofer IIS/EAS in Dresden sollen folgende Teilaufgaben zur Fortführung der Forschungsarbeit in den oben genannten Bereichen bearbeitet werden

- Erstellung eines BRICK-Modells der Anlage (Zentralgerät, Verteilung, Raumübergabe), ggf. Nutzung von BRICK+ und/oder der TUBES Ontologie
- Erstellung und Plausibilisierung von Modelica-Modellen
  - der Luftaufbereitung im Zentralgeräts
  - Vereinfachtes Modell des Verteilsystems
  - Vereinfachtes zeitplanbasiertes Modell der Luftzustandsänderung in den angeschlossenen Räumen
- Vergleich des BRICK- und Modelica-Modells in Hinblick auf automatische Überführbarkeit
- Automatisierte Simulationsläufe zur Erzeugung synthetischer Messdaten
- Erweiterung der vorhandenen Messpunkt- und Topologieerkennung aus Messdaten mit Algorithmen des Machine Learning

DRESDEN concept Exzellenz aus Wissenschaft und Kultur

1. Betreuer:	Dipl.-Ing. Elisabeth Eckstädt – Fraunhofer IIS/EAS
2. Betreuer:	Dr. Andreas Wilde – Fraunhofer IIS/EAS
3. Betreuer:	Prof. Dr.-Ing. habil. Joachim Seifert – TU Dresden

1. Prüfer/Gutachter:	Prof. Dr.-Ing. Clemens Felsmann
2. Prüfer/Gutachter:	Prof. Dr.-Ing. habil. Joachim Seifert

Bearbeitungszeit: 19.11.2020 – 20.05.2021

Prof. Dr.-Ing. Clemens Felsmann
Betreuender Hochschullehrer

Originalaufgabenstellung erhalten: 19.11.2020
Datum/Unterschrift

Kurzfassung

Die Anwendung von Anlagensimulationen bei Planungsphasen in der Gebäudetechnik kann einen bedeutenden Beitrag zur Energieeffizienz leisten. Dennoch ist die Erstellung gebäudetechnischer Simulationsmodellen sehr aufwändig, was das Einsetzen von Simulationen erschwert. Durch die Verbreitung von BIM (Building Information Modelling) könnte die Erstellung der Anlagensimulationen vereinfacht werden, da aus BIM-Daten Informationen über die Anlagen entnommen werden können. Dabei sind semantische Modellen vielversprechend. Andererseits gibt es die Möglichkeit semantische Modelle mithilfe von Machine Learning aus Messdaten zu generieren; Anlagensimulationen können dafür synthetische Messdaten erzeugen. Um die Möglichkeit aus semantischen Modellen Simulationsmodelle abzuleiten zu untersuchen, wurden anhand einer Beispielanlage Modelle in der Brick-Metadatenschema und in Modelica erstellt. Die Modelle wurden in Hinblick auf automatische Überführbarkeit untersucht. Es wurden auch unter Verwendung des Modelica-Modells der Beispielanlage synthetische Messdaten erzeugt, und daraus Zeitreihen mit einem vorhandenen Machine Learning Klassifikationsmodell nach Klassen von Datenpunkten klassifiziert; die Leistung des Klassifikators wurde danach bewertet. Bei der Untersuchung der Darstellungen in Brick und Modelica hat sich ergeben, dass eine automatisierte Überführung ohne Änderungen der Modelle schwierig wäre, da die notwendigen Bedingungen nicht erfüllt wurden. Es wurden dazu Verbesserungsvorschläge präsentiert und Empfehlungen für die Brick-Ontologie gemacht. Bei der Bewertung des Klassifikators, haben sich gemischte Ergebnisse ergeben; dabei hat es sich gezeigt, dass die Wetterbedingungen der Trainingsdaten und der klassifizierten Daten einen großen Einfluss auf die Ergebnisse haben.

Abstract

The implementation of simulations in the planning phases of building technology systems can significantly improve energy efficiency in their future operation. Nevertheless, the creation of simulation models of these systems usually requires much time and effort. The increasing use of BIM (Building information modelling) could ease the creation of these simulation models, as the models could be derived from BIM information. Semantic models of building technology systems appear promising in this regard. Apart from that, there is the possibility to create semantic models from measurement data through the use of machine learning algorithms; simulation models could generate synthetic data for this. In order to evaluate the possibility of an automatic conversion between semantic and simulation models, a model in Modelica and another one in the Brick metadata schema were created and examined. An existing building technical system was taken as example for their creation. The Modelica model was used to generate synthetic data, from which timeseries were classified by an existing machine learning classification model into datapoint classes; the performance of the classification was then assessed. The evaluation of the models in Brick and in Modelica revealed that an automatic conversion between them would be difficult to perform if no previous changes to them are made because the needed requirements for the conversion would not be fulfilled. Further possibilities for the Brick ontology and for an automatic conversion were suggested. The evaluation of the classification offered mixed results, it also showed the marked influence of the weather circumstances of the training and classified data on the performance of the classifier.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Laut dem Bericht *Energieeffizienz in Zahlen* (2020) [16] des Bundesministeriums für Wirtschaft und Energie, betrug in 2018 der gebäuderelevante Endenergieverbrauch in Deutschland 33%, ein erheblicher Anteil. Angesichts der aktuellen Klimakrise hat die Bundesregierung in ihrer Energieeffizienzstrategie beschlossen, bis 2050 (gegenüber 2008) diesen Energieverbrauch um 80% zu reduzieren. Bis 2018 war eine Reduktion von 14.4% bereits geschafft.

Die Anwendung von Simulationen bietet eine Möglichkeit für die Reduktion des gebäudetechnischen Energieverbrauchs, wenn sie zeitig in der Planung eingesetzt werden. Sie können dabei verwendet werden um Fehler in Anlagen zu vermeiden, und um das Regelungssystem optimiert auszulegen. Dennoch ist die Erstellung gebäudetechnischer Simulationsmodellen sehr aufwändig, was das Einsetzen der Simulationen erschwert.

Durch die Verbreitung von BIM (Building Information Modelling) könnte die Erstellung der Anlagensimulationen vereinfacht werden, da aus BIM-Daten Informationen über die Anlagen entnommen werden können. Dabei sind semantische Modellen vielversprechend. In dieser Arbeit wird ein Fokus auf das Brick-Schema gelegt, ein Metadaten-Schema mit dem semantische Modelle erzeugt werden können. Es soll anhand eines Beispielsystems die Möglichkeit einer automatischen Überführung zwischen semantischen Modellen aus dem Brick-Schema untersucht, und Simulationsmodellen die in der Modelica-Modellierungssprache erzeugt werden.

Semantische Modelle sind auch nötig, um manche Anwendungen der Gebäudeautomation einsetzen zu können; z.B. bei Anwendung für Monitoring und Fault Detection, die ebenfalls zur Energieeffizienz beitragen.Die Erstellung von semantischen Modellen muss oft von Experten durchgeführt werden, und ist deswegen häufig teuer und zeitaufwändig. Aktuell ist es deshalb von Interesse die Automatisierung dieses Verfahrens zu untersuchen. Dabei ist die Erkennung von Datenpunkten ein wichtiger Schritt, dies wird mit Modellen des maschinellen Lernens gemacht.

Ein Problem dabei ist häufig der Mangel an Daten um die Algorithmen zu trainieren, deswegen ist es ein aktuelles Forschungsthema das Trainieren von Modellen des maschinellen Lernens mit synthetischen Daten. In dieser Arbeit wird die Leistung eines mit realen Daten trainierten Klassifikationsmodell bei synthetischer Simulationsdaten untersucht.

1.2 Zielstellung

Um die Möglichkeit aus semantischen Modellen Simulationsmodelle abzuleiten, werden anhand eines Beispielsystem ein Brick-Modell und ein Modelica-Modell erstellt. Bei diesen Modellen und bei einem Anlagenschema des Beispielsystems werden die Darstellungen der verschiedenen Komponenten des Beispielsystems verglichen, und ihre Beziehungen zueinander untersucht. Es sollen auch unter Verwendung des Modelica-Modells des Beispielsystems synthetische Simulationsdaten erzeugt werden. Die Daten sollen plausibilisiert und formatiert werden. Daraus werden Zeitreihen mit einem Klassifikationsmodell aus maschinellen Lernen von Mertens [24] nach Klassen von Datenpunkten klassifiziert; die Leistung des Klassifikators soll danach bewertet werden.

2 Beschreibung des Beispielsystems

Das in dieser Arbeit betrachtete RLT-System, ist ein Teilsystem des gesamten RLT-Systems des im Bau befindlichen neuen Institutsgebäudes des Institutsteiles EAS (Entwicklung Adaptiver Systeme) des Fraunhofer IIS an der Münchner Straße in Dresden.

Die betrachtete Anlage umfasst den Bereich in dem sich die Küche, der Essraum, und andere verbundene Räume befinden. Diese Räume haben wegen ihrer Funktion, eine vom anderen Teilen der RLT-Anlage des Gebäudes separate Anlage.

Die Raumbezeichnungen der durch die Anlage versorgten Räume sind: Mitarbeiterversorgung, Speiseausgabe, Relaisküche, Geschirrrückgabe und Geschirr- und Topfreinigung. Ein zugehöriges Schaltschema wurde in *draw.io* erstellt [1], und ist in Abbildung 2.1 zu sehen; es wurde aus einem ursprünglichen Schaltschema das von der Firma Klemm Ingenieure [18] gefertigt wurde, abgeleitet.

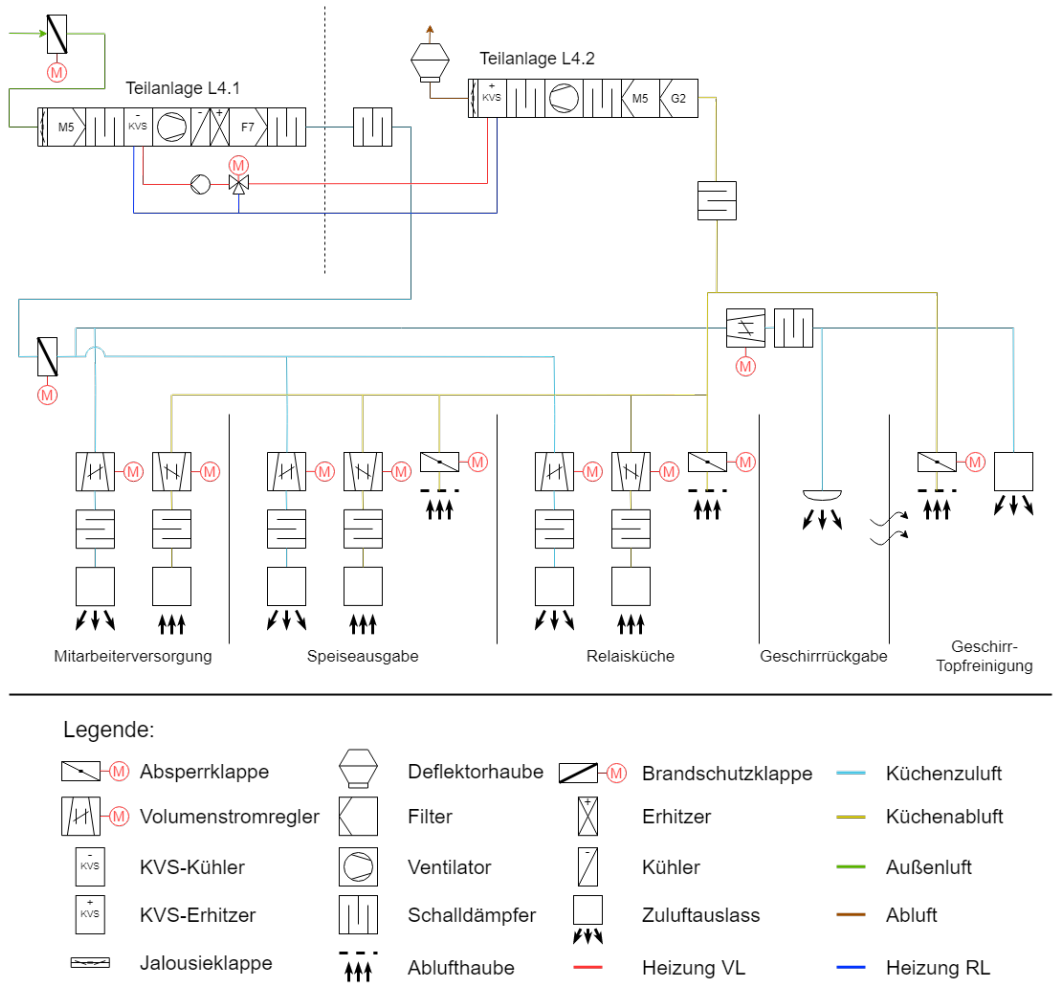


Abb. 2.1: Schaltschema der RLT-Anlage.

Es werden im Schaltschema, 14 Arten von Komponenten dargestellt. Diese sind:

- Ventilator
- Filter
- Schalldämpfer
- KVS-Kühler (KVS: Kreislaufverbundsystem)
- KVS-Erhitzer
- Absperrklappe
- Deflektorhaube
- Jalousieklappe
- Erhitzer
- Kühler
- Volumenstromregler
- Absperrklappe
- Ablufthaube
- Zuluftauslass

Es werden 4 Luftarten und 2 Arten von Wasser-Glykol-Gemisch dargestellt, diese sind im Schaltschema wie folgt bezeichnet:

- Küchenzuluft
- Küchenabluft
- Außenluft
- Abluft
- Heizung VL (VL: Vorlauf)
- Heizung RL (RL: Rücklauf)

Die Gruppen von Komponenten der Teilanlagen L4.1 und L4.2, die die Luft vor- und nachbereiten, befinden sich in zwei Zentralgeräten, welche im Keller und auf dem Dach aufgestellt sind. Das Regelungssystem der Anlage wird in dieser Arbeit nicht betrachtet.

Bei der Vorbereitung der Luft (Teilanlage L4.1), wird die Luft zuerst gefiltert, danach kann Wärme mittels einer Wärmerückgewinnung in Form eines Kreislaufverbundsystems

eingeführt werden. Diese Wärme wird der Küchenabluft entzogen; danach wird die Luft von einem Ventilator weiter befördert. Je nach Bedarf werden durch die Kühler und Erhitzer Wärme zu- oder abgeführt. Die Luft wird wieder gefiltert und ein Schalldämpfer mindert den erzeugten Lärm. Wie dies geschieht kann in Abbildung 2.1 gesehen werden.

In den Kanälen gibt es an zwei Stellen Brandschutzklappen, sie befinden sich vor und nach den Komponenten der Teilanlage L4.1 . Die Luft strömt in die Räume; dies wird durch variable Volumenstromregler geregelt. Die Geschirrrückgabe und die Geschirr- und Topfreinigung haben einen gemeinsamen Kanal für die Einführung der Luft, mit einem einzigen Volumenstromregler und Schalldämpfer für die zwei Räume. Sowohl bei den Einlässen als auch bei den Auslässen der anderen Räume gibt es Schalldämpfer. Volumenstromregler kontrollieren ebenfalls den Austritt der Luft von den Räumen. Die Luft wird danach von der Teilanlage L4.2 nachbereitet; sie wird filtriert, und Wärme abgezogen. Die Wärme zwischen den zwei Wärmeübertragern der Wärmerückgewinnung wird durch ein Wasser-Glykol-Gemisch übertragen. Eine Pumpe und ein Drei-Wege-Ventil zirkulieren das Wasser-Glykol-Gemisch im Kreislaufverbundsystem. Die Luft wird durch eine Ablufthaube in die Umgebung freigesetzt.

3 Brick

3.1 Semantische Datenmodellierung

Ein Semantisches Modell ist eine Darstellung wo ein Objekt durch Instanzen von Klassen die Beziehungen zueinander haben können, beschrieben wird.

Die möglichen Arten von Elementen werden „Klassen" genannt. Die möglichen Beziehungen zwischen Klassen und welche Klassen es geben kann, wird vorher bestimmt. Diese Definition von Klassen, Beziehungen und deren Eigenschaften, heißt Ontologie. Unter Nutzung einer Ontologie kann daraufhin ein semantisches Modell erstellt werden.

Ontologien können für verschiedene Sorten von Darstellungen für verschiedene Zwecke erstellt werden, und enthalten entsprechend Klassen und Eigenschaften, die Objekte im gewünschten Kontext gut darstellen können. In Abbildung 3.1 [19] wird ein Beispiel einer Ontologie gezeigt, mit der Pflanzen und Tiere beschreibt werden können.

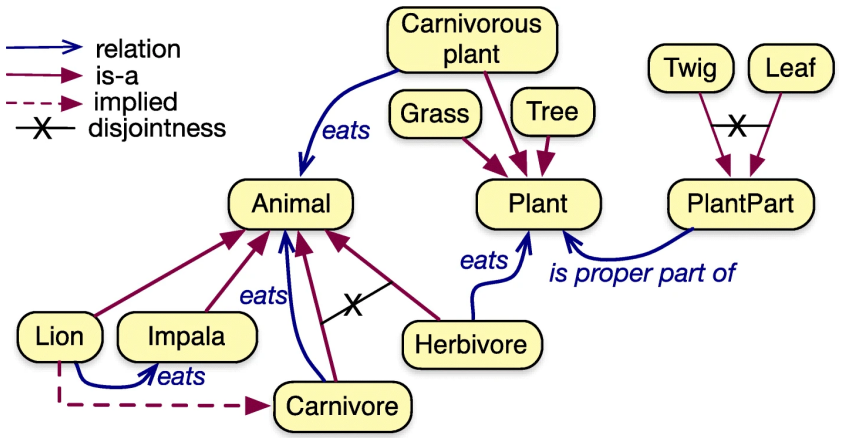


Abb. 3.1: Beispiel einer einfachen Ontologie [19]

Im Rahmen der Gebäudetechnik werden Metadaten-Schemata angewendet, um semantische Modelle der Anlagentechnik und der zugehörenden Sensorik von Gebäuden zu entwerfen. Dabei werden Ontologien benutzt um zu definieren, wie die semantischen Modelle gebaut werden können. Diese semantische Modelle sollen die relevanten Elementen und Beziehungen der Systeme darstellen. Sie können von portablen Softwareanwendungen der Gebäudeautomation verwendet werden, z.B. für Monitoring und Fehlererkennung [3].

Es sind bereits Schemata vorhanden, die dazu dienen können, Metadaten in der Gebäudetechnik darzustellen. Zu diesen Schemata gehören Project Haystack, IFC (Industry Foundation Classes) und Semantic Sensor Web. Laut [4] gibt es Probleme

bei diesen vorhandenen Schemata, da diese die Standardisierung der Darstellungen erschweren.

3.2 Das Brick-Schema

Brick ist ein uniformes Metadaten-Schema, das die Darstellung von Metadaten in Gebäuden dient [3]. Die Motivation hinter der Entwicklung vom Brick-Schema ist die Notwendigkeit einer Lösung für die vorher genannten Probleme bei anderen Metadaten-Schemata in der Gebäudetechnik, wie fehlende Homogenität und Expressivität. Infolge dieser fehlenden Homogenität, können neue regelungstechnische Anwendungen die den Energieverbrauch von Gebäuden reduzieren könnten, nicht angewendet werden [4]. Wegen des erheblich großen Energieverbrauch gebäudetechnischer Anlagen, ist eine Lösung für dieses Problem von Relevanz.

Mithilfe des Brick-Schemas können die Zusammenhänge zwischen Sensorik und Anlagentechnik eines Gebäudes dargestellt werden. Das Brick-Schema definiert eine konkrete Ontologie für Sensoren, Subsysteme, und die Beziehungen zwischen ihnen. Unter Nutzung von diesem Schema, können individuelle Modelle für Gebäude erstellt werden, die ihren zugehörigen gebäudetechnischen Anlagen darstellen. Elemente, die in einem Brick-Modell dargestellt werden, können zu verschiedenen Klassen gehören (z.B. Compressor, Floor, etc.). Brick-Klassen können in 5 wichtige Gruppen eingeteilt werden. Diese werden in Tabelle 3.1 erwähnt und beschrieben.

Tabelle 3.1: Brick-Hauptklassen

Brick-Hauptklasse	Beschreibung
Equipment	physische Komponente einer Anlage
Location	Position
Measurable	messbare Substanz
Point	regelungstechnische Datenquelle
System	gebäudetechnische Anlage

Die möglichen Beziehungen zwischen den Elementen in einem Brick-Modell, ggf. mit ihren zugehörigen reziproken Beziehungen, werden in Tabelle 3.2 gezeigt. Bei der Erstellung eines Brick-Modells ist es nicht notwendig Beziehungen zusammen mit ihren entsprechenden Reziproken zu definieren, da ein *semantic reasoner* die reziproken Beziehungen bestimmen könnte wenn dies notwendig wäre (z.B. bei der Nutzung eines Brick-Modells durch eine Softwareanwendung).

In Abbildung 3.2 [10], wird als Beispiel ein Brick-Modell gezeigt, wo eine gebäudetechnische Anlage charakterisiert wird. Es sind dort 3 Hauptklassen und Arten von Brick-

Tabelle 3.2: Brick-Beziehungen

Klasse	reziproke Klasse
controls	isControlledBy
feeds	isFedBy
hasAssociatedTag	isAssociatedWith
hasLocation	isLocationOf
hasPart	isPartOf
hasPoint	isPointOf
hasTag	isTagOf
measures	isMeasuredBy
regulates	isRegulatedby
hasAddress	-
hasInputSubstance	-
hasOutputSubstance	-
hasTimeseriesId	-
hasUnit	-

Beziehungen vertreten, nämlich *Point*, *Location* und *Equipment*; und *hasPart*,*hasPoint* und *feeds*.

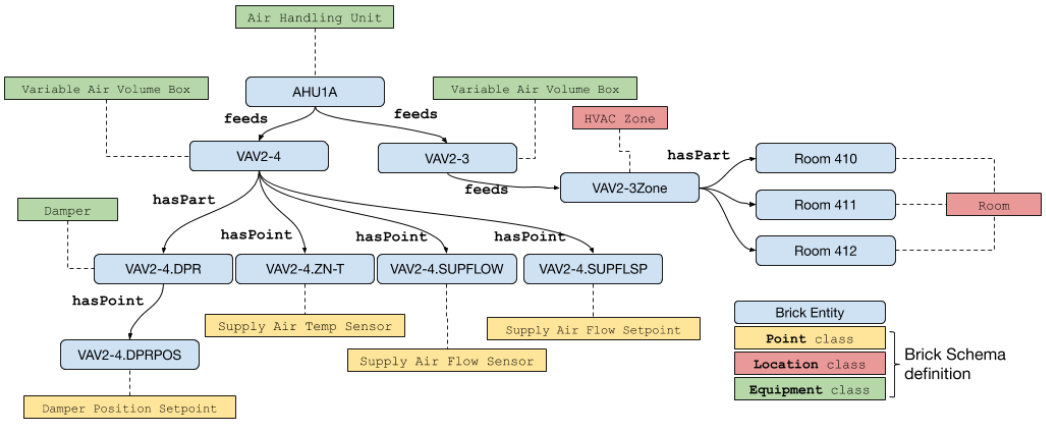


Abb. 3.2: Beispiel eines Brick-Modells [10]

3.3 Erstellung des Brick-Modells

Als Grundlage für die Erstellung eines Brick-Modells in dieser Arbeit wurde ein vorhandenes Brick-Modell einer RLT-Anlage mit zugehörigem Schaltschema von [35] genommen. Außerdem diente als Grundlage das Jupyter-Notebook des Buildsys Brick Tutorials [21].Es wird dort erläutert wie es unter Nutzung der Python-Bibliothek rdflib [5], ein Brick-Modell erzeugt werden kann. Für die Visualisierung der Brick-Modelle und deren Ontologien wurden die Anwendungen *BrickStudio* [8], und *Brick TTL Viewer* [9] verwendet. In dieser Arbeit wurde die Version v1.2 vom Brick-Schema benutzt.

Bei der Erstellung eines Brick-Modells ist es variabel wie viel Detail erfasst wird. Die Positionierung der Elemente (z.B. Raum, Etage), so wie deren Zugehörigkeit zu besonderen funktionalen Gruppen in der Anlage (z.B. Lüftungsanlage, Wärmerückgewinnung) können beschrieben werden. In dieser Arbeit werden Angaben zur Zugehörigkeit von Elementen zu funktionalen Gruppen und Lagen nicht betrachtet, weil die physischen Komponenten der Anlage im Fokus stehen.

Um zu bestimmen, welchen Klassen die einzigen Komponenten der RLT-Anlage gehören, wurde eine Tabelle mit den Komponenten der Anlage und deren Brick-Klassen erstellt (Tabelle 3.3).

Tabelle 3.3: Komponenten im Schaltschema und entsprechende Brick-Klassen

Bezeichnung im Schaltschema	**Brick-Klasse**
Ventilator	Supply Fan
	Exhaust Fan
Pumpe	Water Pump
Filter	Filter
variabler Volumenstromregler	VAV
Jalousieklappe	Damper
Absperrklappe mit Motor	
Brandschutzklappe mit Motor	
Schalldämpfer	Louver
Tellerventil	
Abluftauslass	
Ablufthaube	
Zuluftauslass	
Deflektorhaube	
Abzweigung	-
Kühler	Cooling Coil
KVS-Cooling	
KVS-Heating	Heating Coil
Kühler	
Dreiwegeventil	Ventil
Raum	HVAC Zone
Küchenzuluft	Supply Air
Küchenabluft	Return Air
Außenluft	Outside Air
Fortluft	Exhaust Air
Heizung VL	Water
Heizung RL	

Es fällt auf, dass manche Komponenten sich nicht unbedingt eindeutig zuordnen lassen. Im realen System, funktioniert die Wärmerückgewinnung mit einem Wasser-Glykol-Gemisch als Medium, aber da es unter den aktuellen Brick-Klassen nicht zu finden ist,

wurde es als Wasser dargestellt (Brick v1.2). Andere Komponenten die im Schaltschema näher definiert sind, können in Brick nur durch eine allgemeinere Klasse dargestellt werden, wie bei der Brick-Klasse *Louver*. Bei manchen Komponenten kann es andersherum passieren; die Brick-Klassen *Supply Fan* und *Exhaust Fan* entsprechen im Schaltschema der Bezeichnung *Ventilator*. Es wäre auch möglich die allgemeiner definiert Brick-Klasse *Fan* zu wählen. Die Brick-Klassen *Supply Fan* und *Exhaust Fan* wurden stattdessen gewählt da sie die Komponenten und ihre Funktion in ihrem Kontext besser beschreiben. Von der Position der Ventilatoren im Schaltschema kann gefolgert werden, welcher Brick-Klasse welches Element entspricht.

Es wurden nach der Bestimmung der Brick-Klassen unter Nutzung der Brick-Beziehungen *feeds/isFedBy*, *hasInputSubstance* und *hasOutputSubstance* die verschiedenen Elementen miteinander verbunden um das Brick-Modell zu bilden. Die Brick-Beziehungen *feeds/isFedBy* verbinden Elementen die ein Medium zwischen einander übertragen, und informiert über die Richtung dieser Übertragung. Die Beziehungen *hasInputSubstance* und *hasOutputSubstance* verbinden eine Komponente einer Anlage und ein Medium, und beschreiben ob dieses Medium in oder aus der Komponente übertragen wird. Dies wurde für die Räumen, Strängen und funktionellen Gruppen (WRG, Teilanlagen L4.1 und L4.2) der Anlage gemacht; daraufhin wurden sie in einem Kreislauf verbunden. Ein Bild des resultierenden Brick-Modells mit Angaben zu den Bereichen ist in Abbildung 3.3 zu sehen. Detailliertere Darstellungen zu den einzelnen Bereichen sind in Abbildungen 3.4, 3.5 und im Abschnitt A des Anhangs zu finden. Alle im Modell anwesenden Klassen, und alle mögliche Beziehungen zwischen den einzelnen Klassen, sind in der Ontologie im Abschnitt A des Anhangs zu sehen.

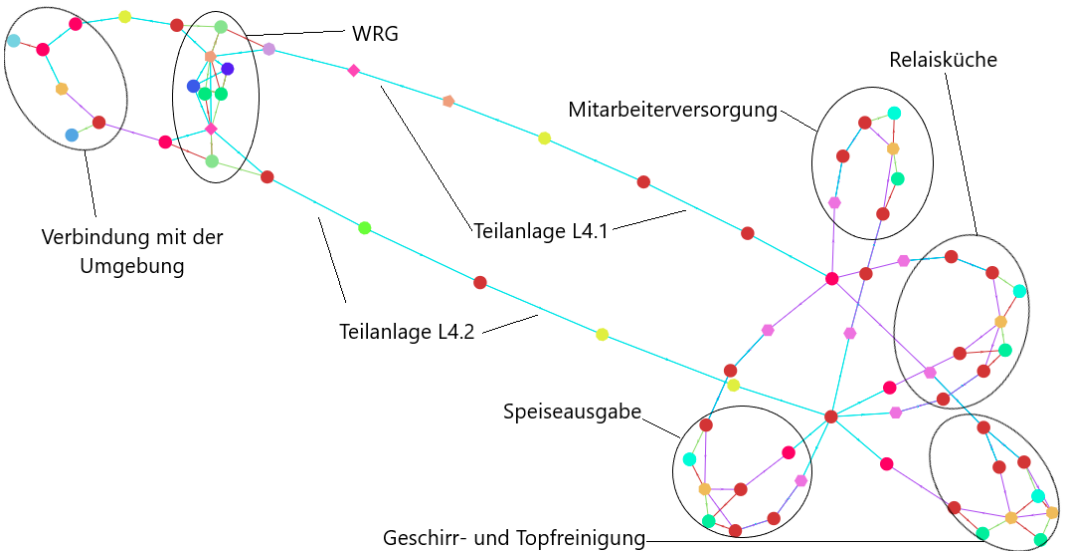


Abb. 3.3: Brick-Modell der RLT-Anlage (Erstellt in *BrickStudio* [8])

Beim bauen des Brick-Modells können einige Aspekte unklar sein, wie z.B. welches Medium durch ein Element fließt, besonders wenn das Element mit mehreren anderen Elementen (3 oder mehr) verbunden ist. Brick bietet keine explizite Weise an, um dies zu definieren. Bei Elementen die mehreren Medien mit anderen Elementen teilen, kann dies problematisch sein, da es nicht eindeutig ist, in welche Richtung welches Medium fließt. Um dies zu Charakterisieren wird hier bei jeder Verbindung darauf hingewiesen welches Medium zwischen den Elementen übertragen wird, indem Parallel zur Beziehung *feeds/isFedBy* die zwei Elementen mit einem dritten Element, welches das Medium repräsentiert, mit den Beziehungen *hasInputSubstance* oder *hasOutputSubstance* (je nach Richtung) verbunden werden. Dafür ist es nötig mehrere Medien zu definieren. In Abbildung 3.4 wird gezeigt wie dies in Brick geschieht. Dort sind einzelne Bausteine zu sehen die durch Pfeile miteinander verbunden sind. Die Pfeile stellen Brick-Beziehungen dar, deren namen auf den Pfeilen zu lesen sind. Die Bausteine (Kreise oder Polygone) sind Instanzen von Brick-Klassen, die Komponenten der Anlage, Medien oder Räume/Areas darstellen. Wenn zwei Bausteine die gleiche Form und Farbe haben, bedeutet dies, dass sie zur selben Klasse gehören. In Abbildung 3.5 ist diese Charakterisierung bei vielen Elementen zu sehen, besonders bei den Wärmeübertragern der Teilanlagen L4.1 und L4.2, wo verschiedenen Medien von oder an anderen Elementen übertragen werden. Detailliertere Darstellungen sind im Abschnitt B des Anhangs zu finden.

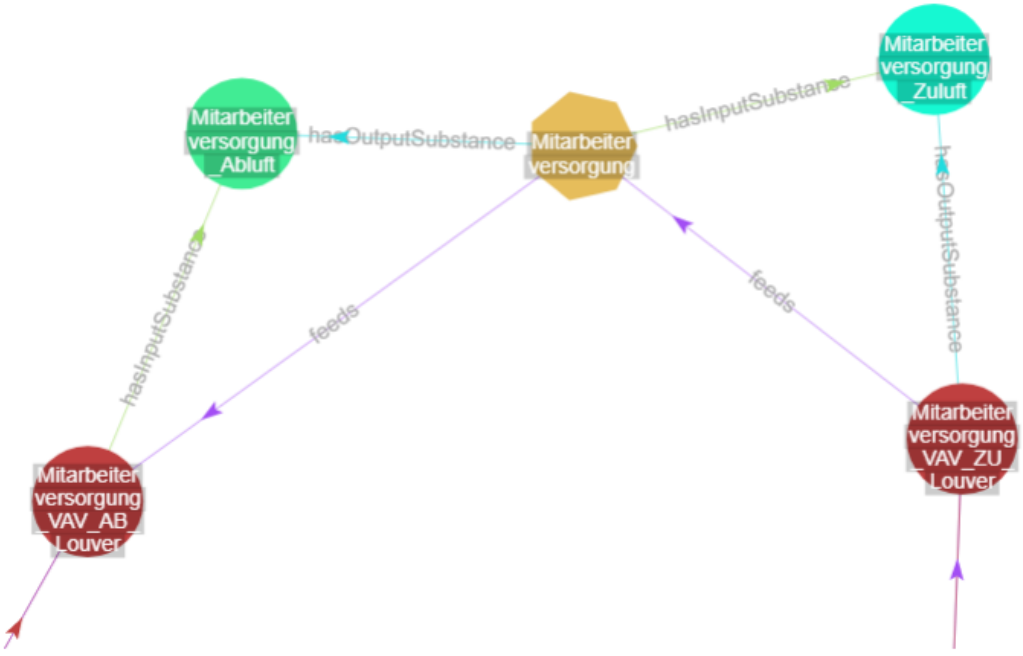


Abb. 3.4: Kennzeichnung der Übertragung eines Mediums zwischen Komponenten in Brick (Erstellt in *BrickStudio* [8])

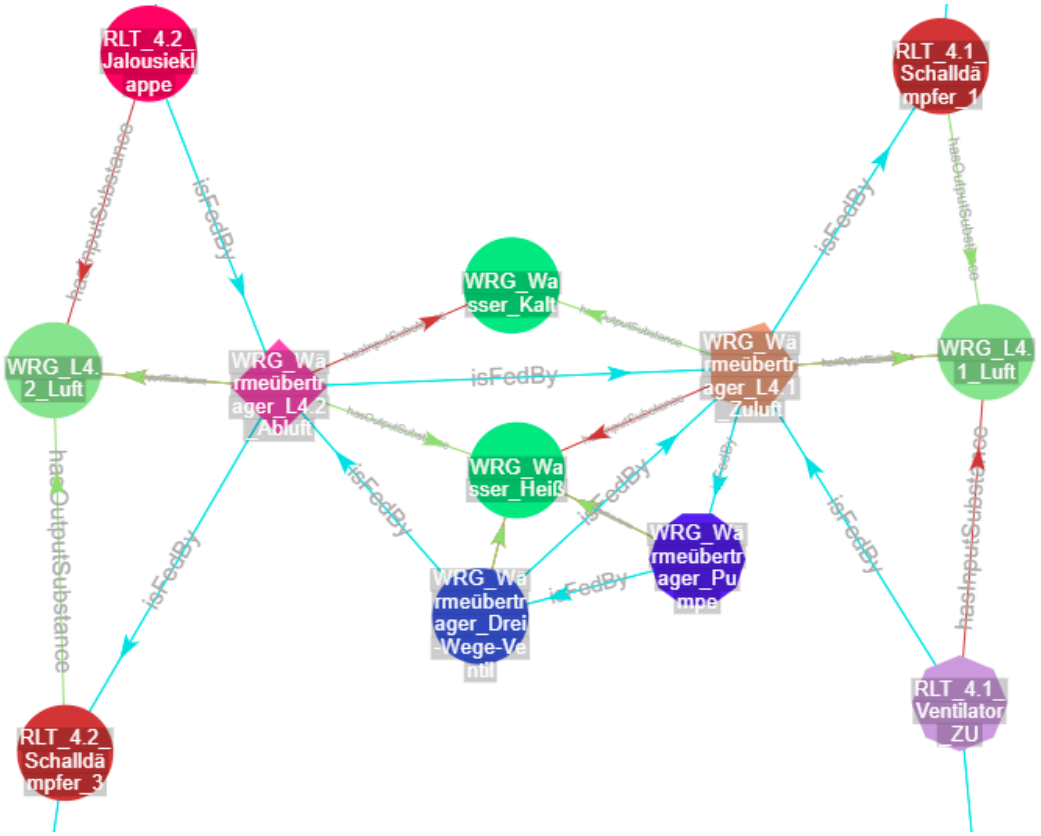


Abb. 3.5: Darstellung der Wärmerückgewinnung (Erstellt in *BrickStudio* [8])

Ein Aspekt von RLT-Anlagen, der sich in Brick nicht darstellen lässt, ist die Positionierung der Abzweigungen von Rohren, weil sie als Elemente nicht dargestellt werden können, da eine entsprechende Klasse fehlt. Aktuell (Brick v1.2) sind außer Rohre auch Kanäle und zugehörige Fittinge bei den Brick-Klassen nicht zu finden.

3.4 Serialisierung

Der erzeugte Code wurde danach in Turtle-Format (.ttl) gespeichert, oder serialisiert. Turtle ist eine Serialisierung; ein Vorgang für die Konversion des Zustands eines Objekts in einen Datenstrom der sofort transportiert werden kann [12]). Die Turtle-Serialisierung wird bevorzugt, da das Turtle-Format von Menschen einfacher zu lesen ist, als andere Serialisierungen, wie z.B. RDF/XML, die schwerer zu verstehen sind wenn sie direkt gelesen werden[23]).

4 Modelica

4.1 Die Modelica-Modellierungssprache

Modelica[1] ist eine Modellierungssprache die der Simulation mathematisch-physikalischer Modelle dient [13]. Modelica ist eine akausale Modellierungssprache welche physische Komponenten durch Gleichungen und nicht durch prozeduralen Code beschriebt. Dies erlaubt die Wiederverwendbarkeit der Modelle und macht es unnötig Gleichungen zu ändern um Modelle an Randbedingungen anzupassen [30].

Der Modelica-Code wird mithilfe einer graphischen Oberfläche einer Simulationsumgebung und möglicherweise auch durch direkte Modifizierung des Codes erstellt. Die hier benutzte Simulationsumgebung ist die Software Dymola. Ein C-Compiler übersetzt den Modelica-Code in die Programmiersprache C. Der C-Code enthält eine mathematische Darstellung des Modells in der Form eines Differentialgleichungssystemes. Ein numerischer Löser berechnet danach eine Lösung für das Gleichungssystem.

Der Löser der in dieser Arbeit benutzt wird, ist DASSL (Differential Algebraic System Solver), ein numerischer Löser von impliziten Systemen algebraischer und differentialer Gleichungssysteme [29]. Der Algorithmus löst das System mithilfe von BDF (Backward Differentiation Formula), und bestimmt die Schrittgröße für jeden Berechnungsschritt automatisch.

Die Modelica Standard-Bibliothek (Version 3.2.3) wird als Quelle von Komponentenmodellen benutzt [2]. Andere Bibliotheken die als Quellen benutzt werden sind: AixLib [25], die Modelica Buildings Library [14], und LibEAS [17]; eine interne Bibliothek des Fraunhofer-Instituts IIS. Es wird die Version 3.4 der Modelica Language Specification benutzt.

4.2 Erstellung des Modelica-Modells

Um ein Modelica-Modell der RLT-Anlage zu bauen, muss es entschieden werden welche Komponentenmodelle zur Modellierung der tatsächlichen Komponenten der RLT-Anlage gewählt werden. Die Komponenten und die entsprechenden gewählten Komponentenmodelle sind in Tabelle 4.1 zu sehen.

Modelica-Modelle können wiederum andere Modelica-Modelle enthalten; dies ermöglicht die Verschachtelung mehrerer Modelle. Die Komponenten die Teil der Wärmerückgewinnung sind: (*Pumpe*, *KVS-Cooling*, *KVS-Heating* und *Dreiwegeventil*); wurden durch das

[1] Grundlegende Informationen und Referenzen zu Modelica wurden aus [22] genommen

Tabelle 4.1: Komponenten im Schaltschema und entsprechende Komponentenmodelle in Modelica

Bezeichnung im Schaltschema	Modelica-Modell
Ventilator	Buildings.Fluid.Movers.FlowControlled_m_flow
variabler Volumenstromregler	Buildings.Fluid.Actuators.Valves. TwoWayEqualPercentage
Jalousieklappe	Buildings.Fluid.Actuators.Valves. TwoWayQuickOpening
Absperrklappe mit Motor	
Brandschutzklappe mit Motor	
Filter	Buildings.Fluid.FixedResistances.PressureDrop
Schalldämpfer	
Tellerventil	
Abluftauslass	
Ablufthaube	
Zuluftauslass	
Deflektorhaube	
Kühler	Buildings.Fluid.HeatExchangers. WetCoilCounterFlow
Pumpe	LibEAS.RLT.Bauteile.HeatRecovery
KVS-Cooling	
KVS-Heating	
Dreiwegeventil	
Erhitzer	Buildings.Fluid.HeatExchangers.HeaterCooler_u
Raum	Buildings.Fluid.MixingVolumes.MixingVolume
Abzweigung	AixLib.Fluid.FixedResistances.Junction
Küchenzuluft	AixLib.Media.Air
Küchenabluft	
Außenluft	
Fortluft	
Heizung VL	Buildings.Media.Water
Heizung RL	

Komponentenmodell *LibEAS.RLT.Bauteile.HeatRecovery* dargestellt, welches andere Komponentenmodelle enthält, die ebenfalls zur Darstellung anderer Komponenten dienen. Welche Komponenten diese sind, wird in Tabelle 4.2 gezeigt. Im Komponentenmodell *LibEAS.RLT.Bauteile.HeatRecovery* kann die Wärmerückgewinnung nicht durch ein Dreiwegeventil, sondern durch einen einstellbaren Massenstrom geregelt werden. Die Inhalte des Komponentenmodells sind in Abbildung 4.1 zu sehen.

Tabelle 4.2: Komponenten im Schaltschema und entsprechende Sub-Modelle in *LibEAS.RLT.Bauteile.HeatRecovery*

Bezeichnung im Schaltschema	Modelica-Submodell
Pumpe	Buildings.Fluid.Movers.FlowControlled_m_flow
KVS-Cooling	Buildings.Fluid.HeatExchangers. WetCoilCounterFlow
KVS-Heating	
Dreiwegeventil	-

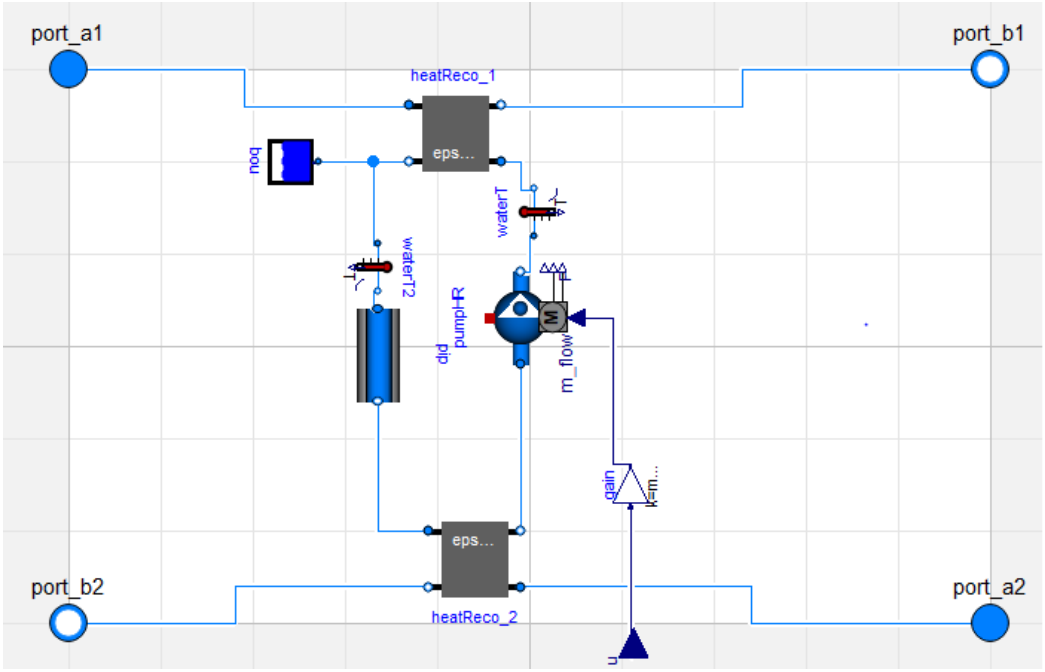


Abb. 4.1: Modelica-Modell der Komponente *Heat Recovery*

Das Komponentenmodell *Buildings.Fluid.Actuators.Valves.TwoWayQuickOpening* wurde zur Darstellung von Komponenten, die wie Ventile funktionieren, aber deren Kennlinien zur Funktion der Anlage nicht kritisch sind, gewählt; sie bleiben normalerweise offen oder geschlossen, die Kennlinie ist dabei eine Potenzfunktion. Andererseits wurde bei der Komponente *variabler Volumenstromregler*, das Modelica-Komponentenmodell *Buildings.Fluid.Actuators.Valves.TwoWayEqualPecrentage* gewählt, da die Kennlinie für die Funktion der Anlage wichtig ist; das dabei gewählte Komponentenmodell soll die Kennlinie eines Volumenstromreglers am besten abbilden können, da sie eine gleichprozentige Ventilkennlinie ist.

Die Komponenten im Schaltschema die durch das Komponentenmodell *Buildings.Fluid.FixedResistances.PressureDrop* repräsentierst sind, sind nicht regelbare Komponenten, und können deswegen durch einen Luftströmungswiderstand dargestellt werden, dessen Wert nur durch die Luftströmung verändert werden kann; dazu wird Formel 4.1 benutzt. Die Bezeichnungen der Komponenten im Schaltschema, die in Tabelle 4.1 der Komponente *Buildings.Fluid.FixedResistances.PressureDrop* entsprechen, sind zu spezifisch und in den verwendeten Modelica-Bibliotheken durch keine vorhandenen Modelle genauer dargestellt. Die Komponenten in Tabelle 4.1 sind nicht die einzigen Sorten von Komponenten die verwendet wurden, sondern nur die, die den Brick-Klassen

zugeordnet werden konnten. Für z.B. die Regelung, wurden auch andere Komponenten benutzt.

$$\dot{m} = k\sqrt{\Delta p} \tag{4.1}$$

In Abbildung 4.2 kann gesehen werden, wie das Modelica-Modell aussieht, nachdem die gewählten Komponentenmodelle miteinander verbunden werden. Zu dieser Abbildung wurde eine Zeichenerklärung erstellt; sie ist im Abschnitt C des Anhangs zu finden. Sie enthält alle Modelica-Komponentenmodelle die hier zur automatischen Überführung relevant sind, außer Medien.

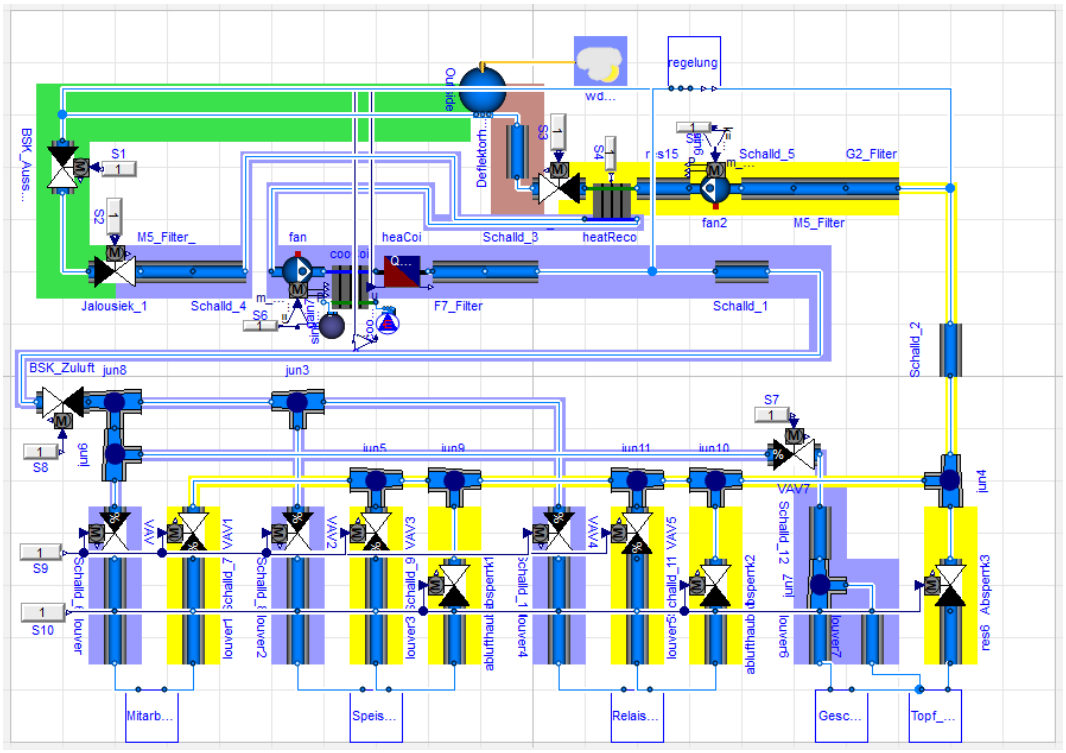


Abb. 4.2: Resultierendes Modelica-Modell

In den Räumen des Gebäudes wird auch der Wärme- und Feuchteeintrag abgebildet. Dabei kommt ein stark vereinfachtes zeitplanbasiertes Modell zur Anwendung. Dafür wurde ein Submodell erstellt, welches das tatsächliche Komponentenmodell das einen Raum darstellt enthält; dabei sind auch andere Komponenten die die Simulation der Wärme- und Feuchteeintrag ermöglichen. Das Ergebnis ist in Abbildung 4.3 zu sehen. Nach [27] und [32] wurde angenommen, dass der Wärme- und Feuchteeintrag jeweils 100W und 90 g/h pro Person wäre. Wären an einem Zeitpunkt z.B. 10 Personen in der Mitarbeiterversorgung, dann wären die gesamten Wärme- und Feuchteeinträge 1000W und 0.9 kg/s. Diese Parameter wurden im Submodell gespeichert. Der

Wärme- und Feuchteeintrag erfolgt nur von Montag bis Freitag, von 12 Uhr bis 15 Uhr.

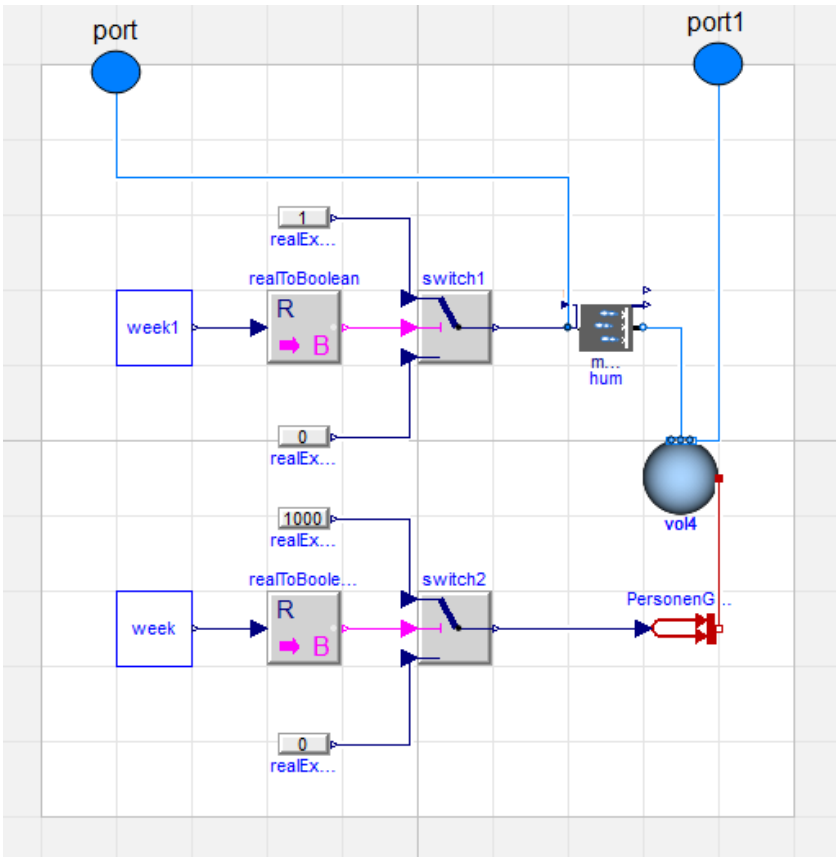


Abb. 4.3: Modelica-Submodell für einen Raum

Für die Regelung der Lüftungsanlage in Modelica, wird ähnlich wie bei den Räumen ein Submodell erstellt. Trotz der fehlenden Darstellung der Regelung in Brick, ist es hier notwendig eine funktionierende Regelung für das Modelica-Modell zu erstellen, da sonst keine sinnvolle Simulationsdaten erzeugt werden können. Die entworfene Regelung entspricht nicht der Realität, aber soll plausibel sein. Das Regelungssystem wird in Abbildung 4.4 gezeigt. Die Regelung wird so ausgelegt, damit die Zulufttemperatur möglichst um 20°C bleibt. Die Signale die geregelt werden, sind diejenigen die die Heiz- und Kühlregister steuern. Wenn die Außentemperatur höher als der Sollwert ist, dann wird die Kühlung eingeschaltet, sonst ist das Signal 0. Das gleiche gilt andersherum für die Heizung. Die Signale, welche die Anlagen steuern, kommen beide von PID-Reglern mit einem Sollwert von 20°C. Die Heizung wird auch eingeschaltet wenn die Zuluft eine Temperatur hat die unter 19°C liegt und die Außentemperatur über 20°C liegt. Dieser Fall kann wegen der ungeregelten Wärmerückgewinnung auftreten. In diesem Fall kommt das Signal für die Heizung von einem PID-Regler mit einem Sollwert von 18.75°C.

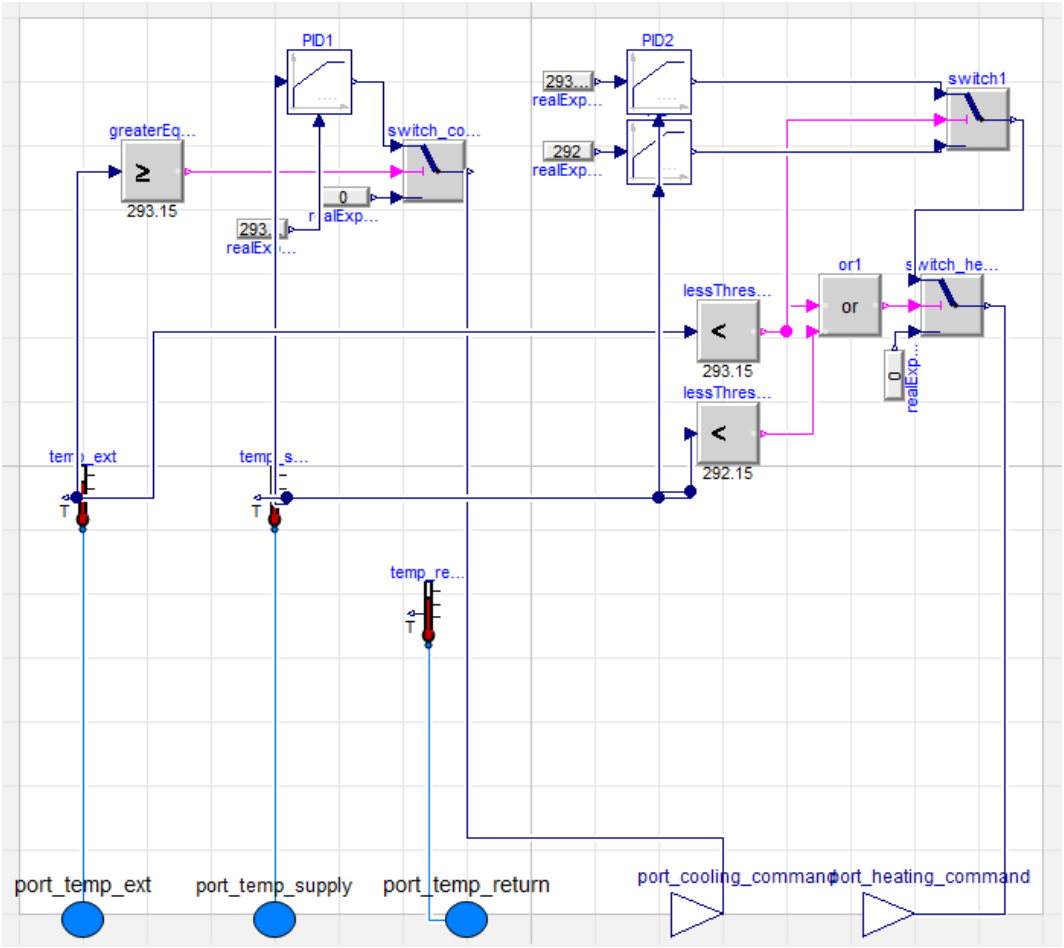


Abb. 4.4: Modelica-Submodell für das Regelungssystem

Die Parameter, die für die Komponenten im Modelica-Modell benutzt werden, entsprechen in der Regel nicht den realen Parametern, sondern andere Parameter die ähnlich oder plausibel sind, da eine realitätsnahe Parametrisierung der Komponentenmodelle der Anlage nicht zu den Zielen dieser Arbeit gehört. Die Parameter Nennheiz- und Nennkühlleistung, Nennmassenstrom und Nenndruckverlust der Kühl- und Heizregister aus der Beispielanlage *Typ1* aus der LibEAS-Bibliothek [17], sollen plausibel sein, und werden hier eingesetzt. Der tatsächliche Nennvolumenstrom der Anlage ist vom ursprünglichen Schaltschema [18] bekannt, und wird im Modell eingesetzt.

Andere wichtige Parameter die hier benutzt werden, sind die Wetterdaten. Diese bestimmen den Zustand der Außenluft, die die Anlage verwendet.Die Wetterdaten die benutzt werden, sind 2 Datensätze aus der Buildings-Bibliothek. Eine Datei enthält Wetterdaten von San Francisco, Kalifornien, in den USA. Die Simulationsergebnisse von Modelica sollen in Abschnitt 7 durch einem Modell des maschinellen Lernens klassifiziert werden. Die Wetterdatei aus San Francisco wurde ausgewählt, da der Klassifikationsmodell mit Gebäudedaten trainiert wurde, die ebenfalls aus Kalifornien kommen. Die andere

Datei enthält Daten aus Chicago, Illinois; auch in den USA. Beide Dateien enthalten Wetterdaten von einem ganzen Jahr.

5 Vergleich von Brick- und Modelica-Modelle in Hinblick auf automatische Überführbarkeit

5.1 Injektivität, Surjektvität und Bijektivität

Für eine mögliche automatische Überführung zwischen den in dieser Arbeit präsentierten Darstellungen der RLT-Anlage (Schaltschema, Brick und Modelica), wäre eine eindeutige und vollständige Zuordnung der Klassen von Elementen in einer Darstellung in anderen Klassen von Elementen in der konvertierten Darstellung wichtig. Da diese Beziehung bei der Überführung eindeutig und komplett sein soll, muss die Beziehung (oder Abbildung) bei der Konversion von einer Darstellung in die andere bijektiv oder surjektiv sein. Dies muss nur in die Richtung der Konversion gelten. Injektivität alleine (ohne Bijektivität) wäre problematisch; da Elemente der Zielgruppe nicht berücksichtigt werden würden. Eine aus einer nur injektiven Überführung resultierenden Darstellung würde nicht alle Klassen der Zielgruppe abbilden können, und wäre dem Original nicht getreu.

Injektivität: eine Beziehung oder Funktion ist injektiv wenn für alle Elementen in der Zielgruppe höchstens ein entsprechendes Element (kein Element ist auch möglich) in der Definitionsgruppe gibt. Hierbei ist die Definitionsgruppe („A" in Abbildung 5.1) die Gruppe der Klassen der Elementen der ursprünglichen Darstellung, und die Zielgruppe(„B" in Abbildung 5.1), die Gruppe der Klassen der Elementen in der konvertierten Darstellung [26].

Surjektivität: eine Beziehung oder Funktion ist surjektiv wenn jedes Element in der Zielgruppe mindestens ein Mal als Funktionswert genommen wird.

Bijektivität: eine Beziehung zwischen zwei Mengen ist bijektiv wenn es für alle Elementen in der Zielgruppe genau ein entsprechendes Element in der Definitionsgruppe gibt. In diesem Fall ist die Funktion injektiv und surjektiv.

In Abbildung 5.1 werden diese Arten von Beziehungen veranschaulicht.

5.2 Vergleichstabellen

In Abschnitt 2, wurde ein Schaltschema der RLT-Anlage präsentiert (Abbildung 2.1), wo alle Komponenten der Anlage, und deren Zusammenhänge zu sehen sind. Mithilfe dieser Informationen wurden in Abschnitt 3.3 jeder Art von Elementen im Schaltschema, eine Brick-Klasse zugeordnet (Tabelle 3.3). In Abschnitt 4.2 wurde mit einer ähnlichen Vorgehensweise gearbeitet, jedoch im Rahmen der Erstellung eines Modelica-Modells; in Tabelle 4.1 sind alle Arten von Elementen des Schaltschema mit den entsprechenden Modelica-Komponentenmodellen zu sehen.

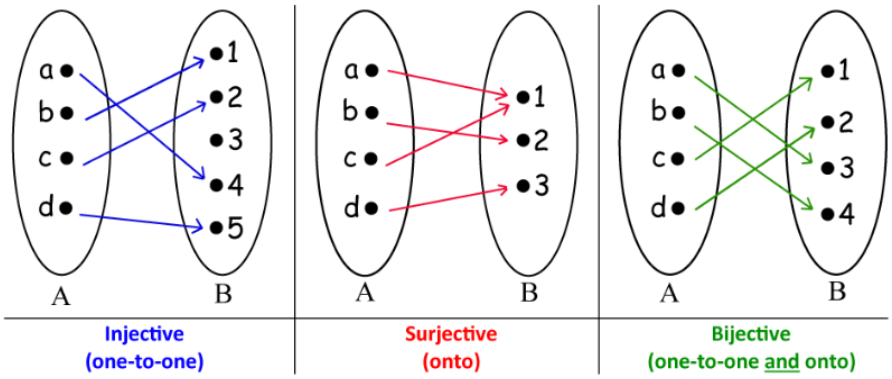


Abb. 5.1: Injektivität, Surjektivität und Bijektivität [6]

5.2.1 Tabelle 5.1

Mithilfe dieser Informationen wird Tabelle 5.1 zum Veranschaulichen der Übereinstimmung der gewählten Modelica-Komponentenmodelle und Brick-Klassen erstellt.

In der Spalte des Schaltschemas gibt es mehr Elemente als in der Brick-Spalte, und es können alle Elementen des Schaltschema eindeutig einer Brick-Klasse zugeordnet werden; mit der Ausnahme der Elementen mit den Bezeichnungen *Ventilator* und *Abzweigung*. Für die Bezeichnung *Abzweigung* gibt es keine entsprechende Brick-Klasse. Bei der Bezeichnung *Ventilator*, gibt es zwei Möglichkeiten bei den Brick-Klassen; *Supply Fan* und *Return Fan*. Diese Klassen wurden gewählt, da sie genauer bezeichnen können, was die Aufgabe der Komponenten in der Anlage ist. Zusätzlich gibt es die Brick-Klasse *Fan*, die beide Ventilatoren bezeichnen kann. Würde man in Brick die allgemeine Klasse *Fan* benutzen, könnte die Spalte des Schaltschemas in Bezug auf die Brick-Klassen bis auf *Ventilator* und *Abzweigung* surjektiv sein. Diese Surjektivität würde als Folge haben, dass wenn ein wie in dieser Arbeit benutzten Schaltschema maschinenlesbar wäre, sich eine automatische Überführbarkeit vereinfachen würde. Dabei wäre es notwendig die Komponenten mit der Bezeichnung *Abzweigung* wegzulassen, oder die Brick-Ontologie zu erweitern.

Andersherum, d.h. von Brick zum Schaltschema wäre eine automatische Überführbarkeit komplizierter, da es z.B. nicht klar wäre, welche Bezeichnung im Schaltschema eine Komponente in einem Brick-Modell der Klasse *Louver* hätte. Bei anderen Brick-Klassen würde es Analog zum selben Problem kommen. Eine mögliche Lösung wäre dabei die Verwendung der Brick-Klasse *Tag* und der Brick-Beziehung *hasTag*, die benutzt werden können um zusätzliche Informationen zu geben. Es könnte bei der Erstellung eines Brick-Modells angegeben werden, welche Bezeichnung die entsprechende Komponente im Schaltschema hat; aber es wäre wahrscheinlich notwendig nicht standardisierte Praktiken zu benutzen, was die Standardisierung vom Brick-Schema schädigen würde. Die Abzweigungen wären da auch noch ein Problem.

Tabelle 5.1: Vergleich der gewählten Modelica-Komponenten und Brick-Klassen

Schaltschema	Brick-Klasse	Modelica-Modell
Ventilator	Supply Fan	Buildings.Fluid.Movers. FlowControlled_m_flow
	Exhaust Fan	
Volumenstromregler	VAV	Buildings.Fluid.Actuators.Valves. TwoWayEqualPercentage
Jalousieklappe	Damper	Buildings.Fluid.Actuators.Valves. TwoWayQuickOpening
Absperrklappe		
Brandschutzklappe		
Filter	Filter	Buildings.Fluid.FixedResistances.PressureDrop
Schalldämpfer	Louver	
Tellerventil		
Abluftauslass		
Ablufthaube		
Zuluftauslass		
Deflektorhaube		
Kühler	Cooling Coil	Buildings.Fluid.HeatExchangers. WetCoilCounterFlow
KVS-Cooling		LibEAS.RLT.Bauteile.HeatRecovery
Pumpe	Water Pump	
Dreiwegeventil	Ventil	
KVS-Heating	Heating Coil	
Erhitzer		Buildings.Fluid.HeatExchangers. HeaterCooler_u
Raum	HVAC Zone	Buildings.Fluid.MixingVolumes.MixingVolume
Abzweigung	-	AixLib.Fluid.FixedResistances.Junction
Küchenzuluft	Supply Air	AixLib.Media.Air
Küchenabluft	Return Air	
Außenluft	Outside Air	
Fortluft	Exhaust Air	
Heizung VL	Water	Buildings.Media.Water
Heizung RL		

Es ist möglich das Brick-Schema lokal zu erweitern und neue Klassen hinzuzufügen, aber dies würde nicht der Standardisierung der Darstellung von Metadaten dienen, was die Motivation hinter Brick ist. Diese Änderung gelten nur lokal, und die Brick-Ontologie [10] bleibt unverändert. Die Brick-Ontologie wird allmählich von The Brick Consortium, Inc. erweitert. Ein Beitrag dazu ist auch möglich, durch Kontakt mit der Brick Ontology Development Arbeitsgruppe, oder direkt in GitHub können Empfehlungen berücksichtigt werden.

Außer der Überführbarkeit der Bezeichnungen der Komponenten, sind auch die Zusammenhänge zwischen ihnen von Bedeutung. Beim Schaltschema es ist ersichtlich welche Komponenten miteinander verbunden sind, was für ein Medium sie übertragen, und in welcher Richtung es fließt. In Brick kann dies unklar sein, wenn ein Element mehrere

Verbindungen zu anderen Elementen hat. Ein Beispiel dafür ist die Abbildung 3.5, welche die zugehörigen Medien charakterisiert. Ohne die Medien wären die jeweiligen Strömungen nicht eindeutig zu verstehen. Um diese Mehrdeutigkeit zu beseitigen, wurde das in Abschnitt 3.3 präsentierte Vorgehensweise vorgeschlagen, die keine Erweiterung von Brick benötigt und bei der Darstellung der Wärmerückgewinnungssystems eingesetzt wurde. In [33], wurde Brick mit der Klasse *Port* erweitert um dieselben Probleme zu lösen, und die Verbindungen zwischen Elementen besser zu charakterisieren. Jedoch wurde dies in Brick v1.2 bisher nicht umgesetzt.

In Modelica ist es sehr eindeutig wie die Komponenten miteinander Verbunden sind, und welche Medien durch sie fließen, aber die Strömungsrichtung ist im Code des Modells nicht festgestellt. Die Nenndurchflussrichtung kann in der graphischen Oberfläche abgelesen werden, aber nur von den Bildern der Komponenten (z.B. die Richtung der Pfeile auf den Ventilatoren in Abbildung 4.2); trotzdem wird diese Richtung nur bei der Simulation bestimmt.

Bei den Medien in den verschiedenen Spalten in Tabelle 5.2 fällt auf, dass in Modelica Luft und Wasser-Glykol-Gemische jeweils nur eine Komponente haben. Um zu bestimmen, welche Bezeichnung ein Medium in Modelica in einer bestimmten Lage einer anderen Darstellung haben soll, wären Angaben zur Funktion oder Lage notwendig, die im Code normalerweise nicht anwesend sind.

Neben der Brick-Klasse *Water*, gibt es auch näher definierte Klassen für ähnliche Medien. Wenn diese hier angewendet würden, wären die Beziehungen zwischen den Spalten der Brick-Klassen und des Schaltschemas bijektiv, was eine automatische Überführbarkeit ermöglichen würde; wie z.B. die Klassen *Supply Water* und *Return Water* für die Vor- und Rückläufe des Kreislaufverbundsystems im Schaltschema.

Beim Modelica-Komponentenmodell *AixLib.Fluid.FixedResistances.Junction*, ist die entsprechende Zelle in der Brick-Spalte leer, da es in Brick keine passende Klasse dazu gibt. Dieses Modell stellt Abzweigungen der Leitungen dar.

Die Elemente in den Spalten von Brick und vom Schaltschema die dem Komponentenmodell *LibEAS.RLT.Bauteile.HeatRecovery* entsprechen, werden als Gruppe von diesem Modell repräsentiert. Diese Gruppe von Elementen bilden das Wärmerückgewinnungssystem. Im Modell *LibEAS.RLT.Bauteile.HeatRecovery* gibt es wiederum Submodelle die diesen Elementen besser entsprechen.

5.2.2 Tabelle 5.2

In Tabelle 5.2 wird derselbe Vergleich gezeigt wie in Tabelle 5.1, aber mit den Submodellen des Modells *LibEAS.RLT.Bauteile.HeatRecovery*.

Tabelle 5.2: Vergleich der gewählten Modelica-Komponenten und Brick-Klassen mit Submodellen für die Wärmerückgewinnung

Schaltschema	Brick-Klasse	Modelica-Modell
Ventilator	Supply Fan	Buildings.Fluid.Movers. FlowControlled_m_flow
	Exhaust Fan	
Pumpe	Water Pump	
Volumenstromregler	VAV	Buildings.Fluid.Actuators.Valves. TwoWayEqualPercentage
Jalousieklappe	Damper	Buildings.Fluid.Actuators.Valves. TwoWayQuickOpening
Absperrklappe		
Brandschutzklappe		
Filter	Filter	Buildings.Fluid.FixedResistances.PressureDrop
Schalldämpfer	Louver	
Tellerventil		
Abluftauslass		
Ablufthaube		
Zuluftauslass		
Deflektorhaube		
Kühler	Cooling Coil	Buildings.Fluid.HeatExchangers. WetCoilCounterFlow
KVS-Cooling		
KVS-Heating	Heating Coil	
Erhitzer		Buildings.Fluid.HeatExchangers. HeaterCooler_u
Dreiwegeventil	Ventil	-
Raum	HVAC Zone	Buildings.Fluid.MixingVolumes.MixingVolume
Abzweigung	-	AixLib.Fluid.FixedResistances.Junction
Küchenzuluft	Supply Air	AixLib.Media.Air
Küchenabluft	Return Air	
Außenluft	Outside Air	
Fortluft	Exhaust Air	
Heizung VL	Water	Buildings.Media.Water
Heizung RL		

Es kann außerdem bemerkt werden, dass in Tabelle 5.2 die Spalte des Schaltschemas bezogen auf die Spalte der Modelica-Komponentenmodelle, abgesehen vom Element *Dreiwegeventil* surjektiv ist. Eine automatische Überführung wäre möglich, wenn es in der Modelica-Spalte ein Modell für Dreiwegeventile gäbe.

Obwohl es in der Modelica-Spalte der Tabelle 5.2 kein Modell für ein Dreiwegeventil gibt (da es kein Submodell des Modells *LibEAS.RLT.Bauteile.HeatRecovery* ist), können in der Buildings Library mehrere Modelle für Dreiwegeventile gefunden werden.

Die Beziehung in Tabelle 5.2 der Brick-Spalte bezogen auf die Modelica-Spalte ist weder surjektiv, noch bijektiv. Es sind näher definierte Klassen in der Brick-Ontologie vorhanden; wenn diese angewendet worden wären, könnte die Beziehung surjektiv sein (abgesehen

von *AixLib.Fluid.FixedResistances.Junction*). Andersherum würde die Beziehung nicht ermöglichen Elemente eindeutig zuzuordnen.

5.3 Diskussion

In Tabellen 5.1 und 5.2 ist die Anzahl an Elementen (mögliche Bezeichnungen im Schaltschema) in der Spalte des Schaltschemas am höchsten unter allen Spalten. In der Brick-Spalte gibt es weniger Elemente. Besonders Elemente, die nicht-regelbare Komponenten beschreiben, kommen selten vor, z.B. bei der Klasse *Louver*. Bei regelbaren Komponenten gibt es mehr Möglichkeiten, um mehr Komponenten genauer zu beschreiben, z.B. bei den Klassen *Supply Air* und *Return Air*. Dies hat auch als folge, dass es keine Brick-Klasse für das Modell *AixLib.Fluid.FixedResistances.Junction* gibt. Es gibt auch kein Modell für Rohrleitungen in Brick.

Dies Passiert weil der Zweck von Brick hauptsächlich Metadaten bezüglich der Sensorik darzustellen ist. Deswegen sind in der Brick-Ontologie hauptsächlich Komponenten von RLT-Anlagen repräsentiert, die Daten erzeugen (Sensorik) oder die durch Signale gesteuert werden.

Das ist bei Modelica nicht der Fall, da Modelica Simulationen erzeugt, auch von Komponenten die nicht durch elektronische Regelsysteme gesteuert sind. Es gibt trotzdem bei den benutzten Modelica-Bibliotheken wenige spezifische Modelle für manche Komponenten, wie bei denen die durch das Modell Buildings.Fluid.FixedResistances.PressureDrop repräsentiert werden.

Es soll kompliziert sein, eine automatische Überführung von vorhandenen Modelica-Modellen und maschinenlesbaren Schaltschemata in Brick-Modellen durchzuführen (mit dem vorhandenen Brick-Ontologie) oder andersherum. Die oben genannten Strategien könnten verwendet werden um eine automatische Überführung zu ermöglichen. Die Darstellungen müssten bewusst erzeugt werden um die Überführbarkeit zu gewähren, was dazu führen könnte, dass die Modelle nicht mehr Standardisiert bleiben und dritten Anwendern unklar erscheinen.

6 Erzeugung synthetischer Messdaten

6.1 Zielsetzung

Die synthetischen Simulationsdaten wurden erzeugt um sie mit einem von Mertens [24] entwickelten Modell des maschinellen Lernens nach Brick-Klassen zu klassifizieren. Die in den Simulationsdaten anwesenden und dem ML-Modell vorhandenen Brick-Klassen sind: *Cooling Valve Command* (ein Signal); *Outside Air Temperature Sensor*, *Return Air Temperature Sensor*, und *Supply Air Temperature Sensor* (Temperaturen). Mit den Ergebnissen der Klassifikation wurde dann die Qualität der Messdatenerkennung des Klassifikators bei synthetischen Daten untersucht.

6.2 Simulationsläufe

Für die Erstellung von synthetischen Messdaten wurde das Simulationsmodell aus Abschnitt 4 benutzt. Dieses Modell kann mit verschiedenen Wetterdaten arbeiten die im *.mos*-Format gespeichert sind. Somit kann das Verhalten vom System in verschiedenen Klimaregionen simuliert, und diverse synthetischen Daten erzeugt werden. Die Wetterdaten können als ein Parameter betrachtet werden.

Es wurden zwei Gruppen von synthetischen Daten durch zwei Modelica-Simulationen erstellt, eine mit Wetterdaten aus Chicago, Illinois (Köppen-Klimaklassifikation: Dfa-Feuchtes Kontinentalklima) und eine andere mit Daten aus San Francisco, Kalifornien (Köppen-Klimaklassifikation: Csc-Mediterranes Klima); die Köppen-Klimaklassifikation für Dresden ist Cfb-Ozeanisches Klima [7]. Diese Wetterdaten wurden jeweils über ein Jahr gemessen; deswegen betragen die entsprechenden Modelica-Simulationen auch ein Jahr.

6.3 Vorbereitung der Daten

Die Ergebnisse der Simulationen in Dymola werden in .mat Format gespeichert. In diesem Format sind die Werte der Messungen in Spalten gespeichert, deren Namen jeweils den Namen der Komponente in Dymola entsprechen. Die Zeitspalte enthält die Zeit der Aufnahme der Werte.

Der Klassifikator wurde so ausgelegt, dass die eingelesenen Zeitreihen 24 Stunden betragen und einen zeitlichen Abstand von 15 Minuten zwischen den einzelnen Werten haben [24]. Beim Simulationsergebnis von Dymola beträgt die Abtastperiode zwischen den einzelnen Werten der Zeitreihen eine Minute. Um dies anzupassen, wurde die

	Simulation.regelung.switch_cooling_command.y	Simulation.regelung.temp_ext.T	Simulation.regelung.temp_return.T	Simulation.regelung.temp_supply.T	Time
0	0.0	280.350006	293.149994	293.149994	1.577837e+09
1	0.0	280.350006	293.149750	293.150116	1.577837e+09
2	0.0	280.350006	293.149750	293.150116	1.577837e+09
3	0.0	280.350006	293.149567	293.150177	1.577837e+09
4	0.0	280.350006	293.149567	293.150177	1.577837e+09
...	...	...	...	...	...
530703	0.0	284.510010	293.149994	293.149994	1.609373e+09
530704	0.0	284.523346	293.149994	293.149994	1.609373e+09

Abb. 6.1: Rohe Daten aus Dymola.

Methode *pandas.DataFrame.resample* angewendet, um ein Resampling auf 15 Minuten auszuführen [28]. Die Simulationsdaten wurden in Zeitreihen von 24 Stunden aufgeteilt.

Um die Ergebnisse aus Dymola (Abbildung 6.1) für den Klassifikator lesbar zu machen, müssen sie in ein *pandas.DataFrame* gespeichert, und in das Format von MortarData (Abbildung 6.2) umgeformt werden.

	time	id_day	name	id_unique	classes	value
0	2018-10-16 00:00:00+00:00	289	GBSF.AHU.AHU01.Cooling_Valve_Output	289_GBSF.AHU.AHU01.Cooling_Valve_Output	Cooling_Valve_Command	16.000000
1	2018-10-16 00:15:00+00:00	289	GBSF.AHU.AHU01.Cooling_Valve_Output	289_GBSF.AHU.AHU01.Cooling_Valve_Output	Cooling_Valve_Command	18.000000
2	2018-10-16 00:30:00+00:00	289	GBSF.AHU.AHU01.Cooling_Valve_Output	289_GBSF.AHU.AHU01.Cooling_Valve_Output	Cooling_Valve_Command	17.000000
3	2018-10-16 00:45:00+00:00	289	GBSF.AHU.AHU01.Cooling_Valve_Output	289_GBSF.AHU.AHU01.Cooling_Valve_Output	Cooling_Valve_Command	8.000000
4	2018-10-16 01:00:00+00:00	289	GBSF.AHU.AHU01.Cooling_Valve_Output	289_GBSF.AHU.AHU01.Cooling_Valve_Output	Cooling_Valve_Command	0.000000
...	...	...	...	...	...	...
5128603	2018-10-28 22:45:00+00:00	301	CHEMX.CHW.BUILD_DP	301_CHEMX.CHW.BUILD_DP	Differential_Pressure_Sensor	70671.257812
5128604	2018-10-28 23:00:00+00:00	301	CHEMX.CHW.BUILD_DP	301_CHEMX.CHW.BUILD_DP	Differential_Pressure_Sensor	69430.203125

Abb. 6.2: Daten aus MortarData.

Im Gegenteil zur Struktur der Daten in den *.mat* Dateien von Dymola, belegt ein Wert einer Zeitreihe jeweils eine komplette Zeile, zusammen mit Metadaten wie Zeit, Identifikator der Zeitreihe, etc. Bei den Ergebnissen von Dymola sind in einer Zeile alle Werte von verschiedenen Komponenten in nebeneinander liegenden Spalten enthalten.

Um die Umformung durchzuführen, müssen von der *.mat* Datei die Spalten von Werten der verschiedenen Komponenten, grob gesagt aufeinander gestapelt, und in eine neue Matrix gespeichert werden.

Weitere nötige Änderungen sind die Zusätze von Spalten mit dem entsprechenden Tag im Jahr (*id_ day*), welcher Klasse die Zeitreihe gehört, Platz des Wertes innerhalb der Zeitreihe, und Name der Zeitreihe (ist in diesem Fall *id_ unique* gleich).

Die Simulationen wurden so gemacht, dass jede Minute alle Werte der Variablen im System berechnet und gespeichert wurden. Die Ergebnisse der Simulationen haben jeweils über 530000 Einträge für jede Variable. Nach dem Resampling, ist die Anzahl dieser Einträge auf rund 35000 gesunken. Da diese Einträge in Tage unterteilt werden, gibt es für jede Brick-Klasse 365 Zeitreihen in den Daten der jeweiligen Simulation,

	time	id_day	name	id_unique	classes	value	t
0	2020-01-01 00:00:00	1	Simulation.regelung.switch_cooling_command.y1	Simulation.regelung.switch_cooling_command.y1	Cooling_Valve_Command	0.000000	0
1	2020-01-01 00:15:00	1	Simulation.regelung.switch_cooling_command.y1	Simulation.regelung.switch_cooling_command.y1	Cooling_Valve_Command	0.000000	1
2	2020-01-01 00:30:00	1	Simulation.regelung.switch_cooling_command.y1	Simulation.regelung.switch_cooling_command.y1	Cooling_Valve_Command	0.000000	2
3	2020-01-01 00:45:00	1	Simulation.regelung.switch_cooling_command.y1	Simulation.regelung.switch_cooling_command.y1	Cooling_Valve_Command	0.000000	3
4	2020-01-01 01:00:00	1	Simulation.regelung.switch_cooling_command.y1	Simulation.regelung.switch_cooling_command.y1	Cooling_Valve_Command	0.000000	4
...	...	...	...	...	...	...	...
140159	2020-12-30 23:00:00	365	Simulation.regelung.temp_supply.T365	Simulation.regelung.temp_supply.T365	Supply_Air_Temperature_Sensor	19.999994	92
140160	2020-12-30 23:15:00	365	Simulation.regelung.temp_supply.T365	Simulation.regelung.temp_supply.T365	Supply_Air_Temperature_Sensor	19.999994	93

Abb. 6.3: Umgeformte Daten aus Dymola.

eine Zeitreihe pro Tag im Jahr. Da mit zwei verschiedenen Wetterdatensätzen simuliert wurde existieren zwei derartige Messreihen.

6.4 Plausibilisierung der Simulationsdaten

Als Plausibilitätskontrolle werden zwei Größen überprüft, die sich aus komplexen Zusammenhängen zwischen mehreren Komponenten im Modell ergeben; diese sind *Simulation.regelung.temp_supply.T* und *Simulation.regelung.temp_return.T*. Die entsprechenden Brick-Klassen sind *Supply Air Temperature Sensor* und *Return Air Temperature Sensor*. *Simulation.regelung.temp_supply.T* ist besonders wichtig, da die Regelung sie als Regelgröße benutzt um die Heiz- und Kühlregister zu steuern.

6.4.1 Simulation mit Wetterdaten aus Chicago

In Abbildung 6.4 kann gesehen werden, dass der jährliche Verlauf der Zulufttemperatur der Simulation mit Wetterdaten aus Chicago nah am Sollwert (20°C) bleibt. Es sind mehrere Peaks zu sehen, allerdings erreichen die meisten nur (20.88°C) und dauern nur wenige Minuten. Sie werden durch den Wechselbetrieb zwischen Heizen und Kühlen verursacht, und sind hauptsächlich in der wärmeren Zeit des Jahres zu sehen. Die Werte, welche die Zulufttemperatur in der Simulation annimmt, stellen einen plausiblen Wertebereich für eine reale Zuluftemperatur einer RLT-Anlage dar.

Abbildung 6.5 zeigt der Verlauf der Ablufttemperatur. Hier ist die Spannweite und das Auftreten der Peaks größer als in Abbildung 6.4. Sie werden durch die zyklischen Wärmeeinträge (siehe Abschnitt 4.2) verursacht.

6.4.2 Simulation mit Wetterdaten aus San Francisco

Der jährliche Verlauf der Ablufttemperatur in den Simulationsdaten mit Wetterdaten aus San Francisco wird in Abbildung 6.7 gezeigt. Analog zu Abbildung 6.5 ist die

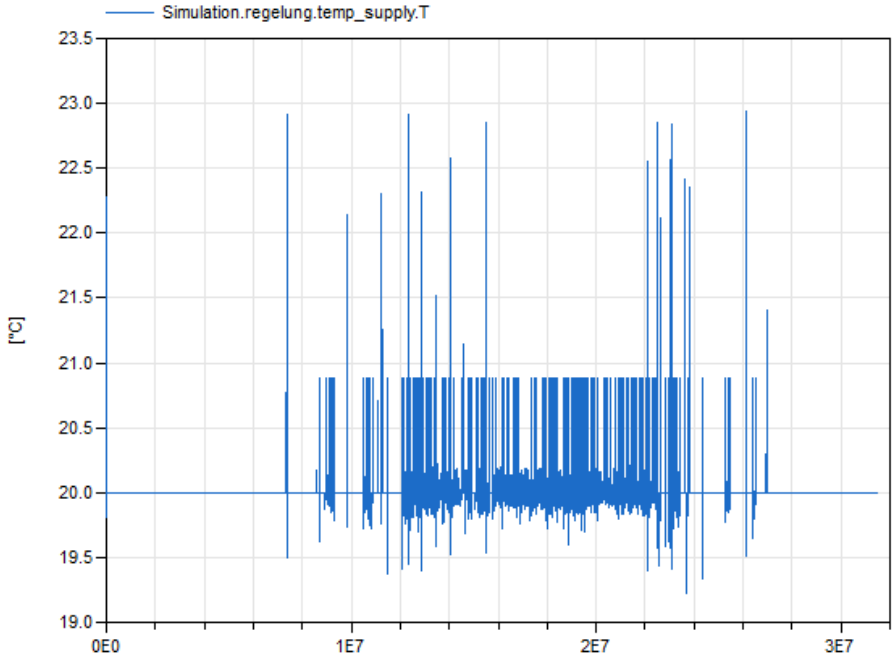


Abb. 6.4: Jährlicher Verlauf der Zuluftemperatur in den Simulationsdaten (Chicago)

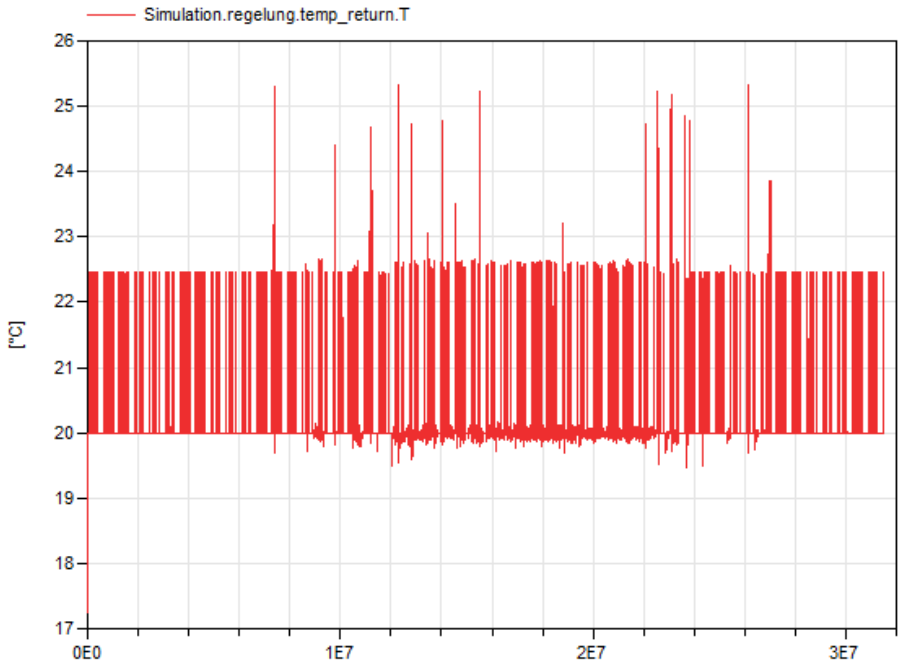


Abb. 6.5: Jährlicher Verlauf der Ablufttemperatur in den Simulationsdaten (Chicago)

Spannweite größer; die zyklischen Änderungen infolge des Wärmeeintrags können hier ebenfalls betrachtet werden.

In Abbildung 6.6 wird der jährliche Verlauf der Zulufttemperatur gezeigt. Es kann bemerkt werden, dass es mehr Peaks als in Abbildung 6.4 gibt. Dies passiert, da das Klima in San Francisco allgemein wärmer ist, als das in Chicago. Daher tritt der Wechsel zwischen Heizbetrieb und Kühlbetrieb häufiger als in Chicago auf. Die

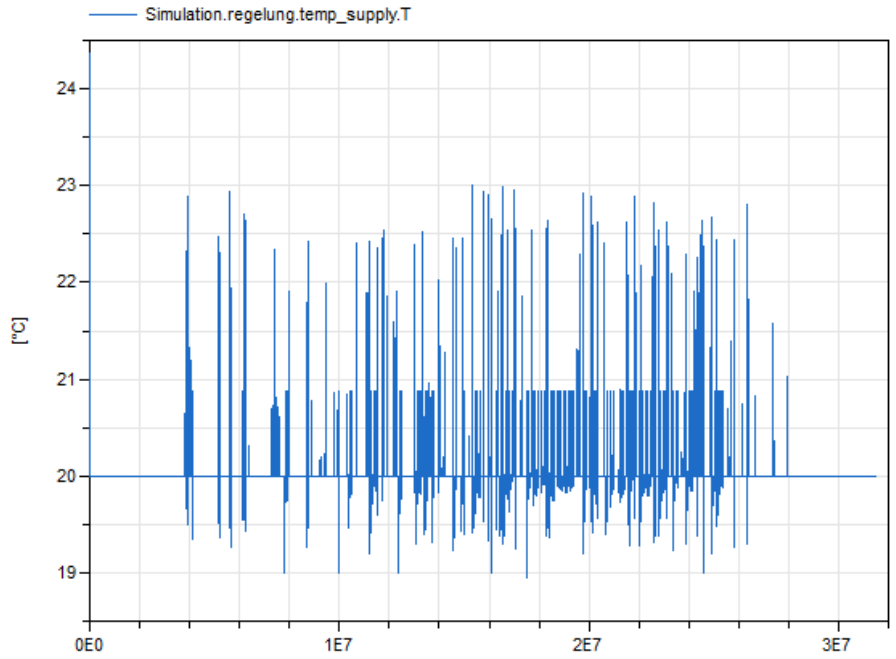


Abb. 6.6: Jährlicher Verlauf der Zuluftemperatur in den Simulationsdaten (San Francisco)

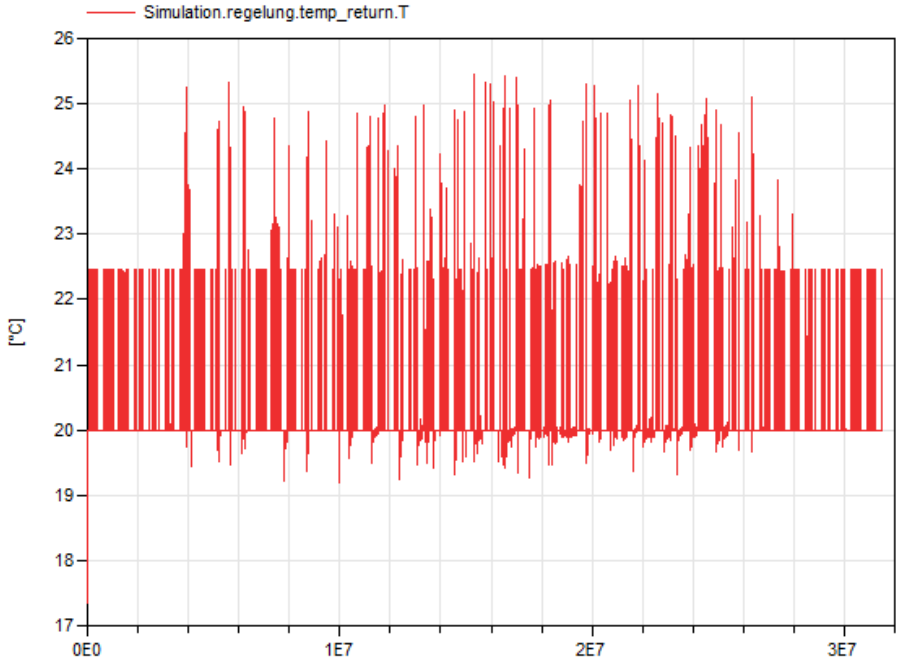


Abb. 6.7: Jährlicher Verlauf der Ablufttemperatur in den Simulationsdaten (San Francisco)

Spannweite der meisten Peaks erreicht wie in Abbildung 6.4 auch 20.88°C, somit bleibt der Verlauf hier auch nah am Sollwert (20°C). Trotz der häufigeren und ausgeprägteren Abweichungen, bleibt die Zulufttemperatur innerhalb der Behaglichkeitszone, in einem plausiblen Wertebereich.

7 Messpunkt- und Topologieerkennung mit Algorithmen des ML

7.1 Maschinelles Lernen

Maschinelles Lernen (ML) beschreibt im Kontext der Informatik die Generierung von Wissen in Form von Regeln oder Algorithmen, aus elektronischen Daten wie Bilder, Zahlen, Kategorien, etc., den sie ausgesetzt werden; wobei die Algorithmen die das Wissen generieren nicht explizit mit diesem Zweck programmiert werden. Das generierte Wissen wird in einem Modell gespeichert, das später angewendet werden kann; diese Modelle bekommen Eingabedaten, und geben Ausgabedaten zurück.

Zwei der wichtigsten Kategorien im maschinellen Lernen sind überwachtes Lernen und unüberwachtes Lernen. Sie unterscheiden sich indem sie Feedback beim Lernen zur Verfügung haben oder nicht. Bei überwachtes Lernen hat der Algorithmus Zugriff auf Feedback das die Ausgabedaten bewertet; dann kann der Algorithmus die erzeugten Regel daran anpassen. Bei unüberwachtes Lernen hat der Algorithmus keinen Zugriff auf Feedback, und sucht nach Mustern und Beziehungen innerhalb der Eingabedaten. Da in dieser Arbeit nur überwachtes Lernen relevant ist, wird nicht auf anderen Kategorien von ML-Algorithmen eingegangen.

Überwachtes Lernen wird oft benutzt um Klassifikationen oder Regressionen zu machen; bei Klassifikatoren wird eine Klassifikationsbezeichnung, und bei Regressionsmodelen eine kontinuierliche Größe vorhergesagt [34]. Ein Beispiel für ein ML-Klassifikationsmodell ist der Algorithmus der Spam unter E-Mails erkennt. Ein Beispiel für ein Regressionmodell ist ein Algorithmus der Preise von Immobilien anhand von anderen Informationen wie Lage, Baujahr, Wohnfläche, etc., ermittelt. Das Modell wird zuerst mit Daten trainiert, die Trainingsdaten genannt werden. Danach wird ein Datensatz (Validierungsdaten) dem Modell als Eingabe gegeben; anhand der Ergebnissen werden noch Parameter angepasst. Ein anderer Datensatz (Testdaten), wird dann benutzt um die Leistung des ML-Modells zu bewerten.

7.2 Beschreibung des benutzten ML-Modells

Das in dieser Arbeit verwendete Klassifikationsmodell wird von Mertens [24] übernommen. Die Arbeit von [24] soll ein Baustein für die automatisierte Erstellung von semantischen Modellen sein, die später das Einsetzen von portablen Softwareanwendungen im Rahmen der Gebäudeautomation ermöglichen sollen. Da wurde die Wahl der Features schon gemacht, und der Klassifikator trainiert. Features sind numerische Merkmale (z.B. Maximum, Minimum, Varianz) die benutzt werden um einzelne Objekte oder Samples (z.B. eine Druckmessung über eine Zeitperiode) zu beschreiben [24]. Das

Klassifikationsverfahren von diesem Modell, ist *ExtraTrees* [31]. Dieser Verfahren ist eine Art des *Random Forest* Klassifikationsverfahrens. Hier werden mehrere Entscheidungsbäume unter Betrachtung der Features erzeugt und danach benutzt. Diese werden unabhängig voneinander und unterschiedlich erzeugt. Jeder Entscheidungsbaum bestimmt in mehreren Stufen durch *wenn-dann*-Regel, welche die Features betrachten, zu welcher Klasse ein betrachtetes Objekt gehört. Die verschiedene Ergebnisse der Entscheidungsbäume werden nach Häufigkeit gewichtet, und die häufigste Klasse als endgültiges Ergebnis genommen [24]. Parameter wie die Wahl der Features werden anhand der Klassifikationsergebnisse von einem Programmierer optimiert.

Dieser Klassifikator hat die Aufgaben, Zeitreihen anhand der enthaltenen Messwerte zu klassifizieren. Dabei soll erkannt werden, ob es sich beispielsweise um einen Temperaturmesswert oder einen Ventilstellwert handelt; diese Information soll in Form einer Brick-Klasse als Ausgabe gegeben werden. Der Klassifikator wurde mit Daten trainiert, die aus dem MortarData-Datenbank kommen [15]. Die Zeitreihen in den Trainingsdaten und in den Daten die klassifiziert werden, entsprechen Signalen von Sensoren oder Befehlen in Komponenten, die eine entsprechende Brick-Klassen haben. Die MortarData-Datenbank enthält eine große Menge an realen RLT-Daten, die aus Gebäuden in Kalifornien kommen.

Die Simulationsergebnisse wurden dementsprechend (wie in Abschnitt 6.3 beschrieben) aufbereitet und fungieren im folgenden als „synthetische Messdaten".

7.3 Verwendete Variablen und Brick-Klassen

Mit der Absicht, die Qualität von Messdatenerkennung von synthetischen Messdaten zu bewerten, wurde der Klassifikator auf die in Modelica erzeugten Daten angewendet.

Es gibt neun Brick-Klassen, die von dem Klassifikator betrachtet werden:

- Cooling Valve Command
- Damper Position Setpoint
- Differential Pressure Sensor
- Economizer Damper Command
- Outside Air Temperature Sensor
- Return Air Temperature Sensor
- Supply Air Flow Sensor
- Supply Air Static Pressure Sensor

- Supply Air Temperature Sensor

In dem vorliegenden Modelica-Modell gibt es für folgende Klassen ein Beispiel:

- Cooling Valve Command
- Outside Air Temperature Sensor
- Return Air Temperature Sensor
- Supply Air Temperature Sensor

Ihre Positionen im Modelica-Modell ist in Abbildung 7.1 dargestellt.

Tabelle 7.1: Gewählte Variablen aus Modelica-Modell und entsprechende Brick-Klassen.

Name der Variable	Brick-Klasse
Simulation.regelung. switch_cooling_command.y	Cooling Valve Command
Simulation.regelung.temp_ext.T	Outside Air Temperature Sensor
Simulation.regelung.temp_return.T	Return Air Temperature Sensor
Simulation.regelung.temp_supply.T	Supply Air Temperature Sensor

Diese Variablen können mit den entsprechenden Brick-Klassen beschrieben werden, da sie genau die Beschreibung der Klassen erfüllen, oder sehr ähnlich sind. Die Brick-Klassen *Outside Air Temperature Sensor*, *Return Air Temperature Sensor*, *Supply Air Temperature Sensor* bezeichnen jeweils Temperaturen von Außenluft, Abluft und Zuluft, genauso wie die Variablen in Tabelle 7.1, außer *Simulation.regelung.switch_cooling_command.y*.

Die Variable *Simulation.regelung.switch_cooling_command.y* stellt den Verlauf des Signals der Steuerung der Kühlung. Im Modelica-Modell wird dadurch ein idealer variabler Massenstrom geregelt. Die Brick-Klasse *Cooling Valve Command* bezeichnet das Steuersignal einer Kühlventil und erfüllt eine analoge Aufgabe.

Die Komponenten im Modelica-Modell die Variablen erzeugen, befinden sich im Submodell „Regelung". Diese Komponenten werden in Abbildung 7.2 gezeigt.

7.4 Ergebnisse der Klassifikation und Diskussion

7.4.1 Bewertungskriterien

Bevor die Ergebnisse bewertet werden, sollen relevante Begriffe zur Bewertung von ML-Modellen eingeführt werden [24].

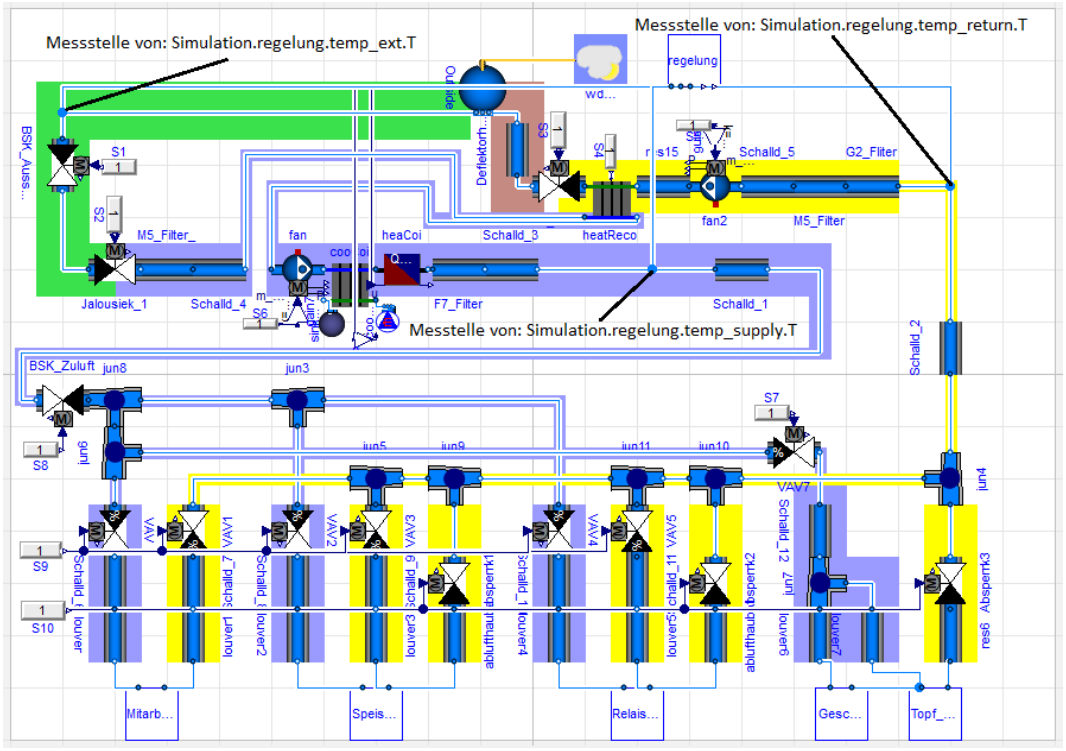


Abb. 7.1: Messstellen der Temperatursensoren.

In Abbildung 7.3 ist eine Konfusionsmatrix für eine Klassifikationsaufgabe mit zwei möglichen Ergebnissen („Positive" und „Negative") gezeigt. Die Konfusionsmatrix dient der Bewertung einer Klassifikationsaufgabe.

Precision (Formel 7.1): quantifiziert die Anzahl der positiven Klassenvorhersagen, die tatsächlich zu der positiven Klasse gehören [24]. Precision besagt wie genau die Auswahl von positiven Vorhersagen ist.

$$precision = \frac{TP}{TP + FP} \tag{7.1}$$

Recall (Formel 7.2): quantifiziert die Anzahl der positiven Klassenvorhersagen an allen positiven Vorhersagen des gesamten Datensatzes [24]. Recall besagt wie komplett die Auswahl von positiven Vorhersagen ist.

$$recall = \frac{TP}{TP + FN} \tag{7.2}$$

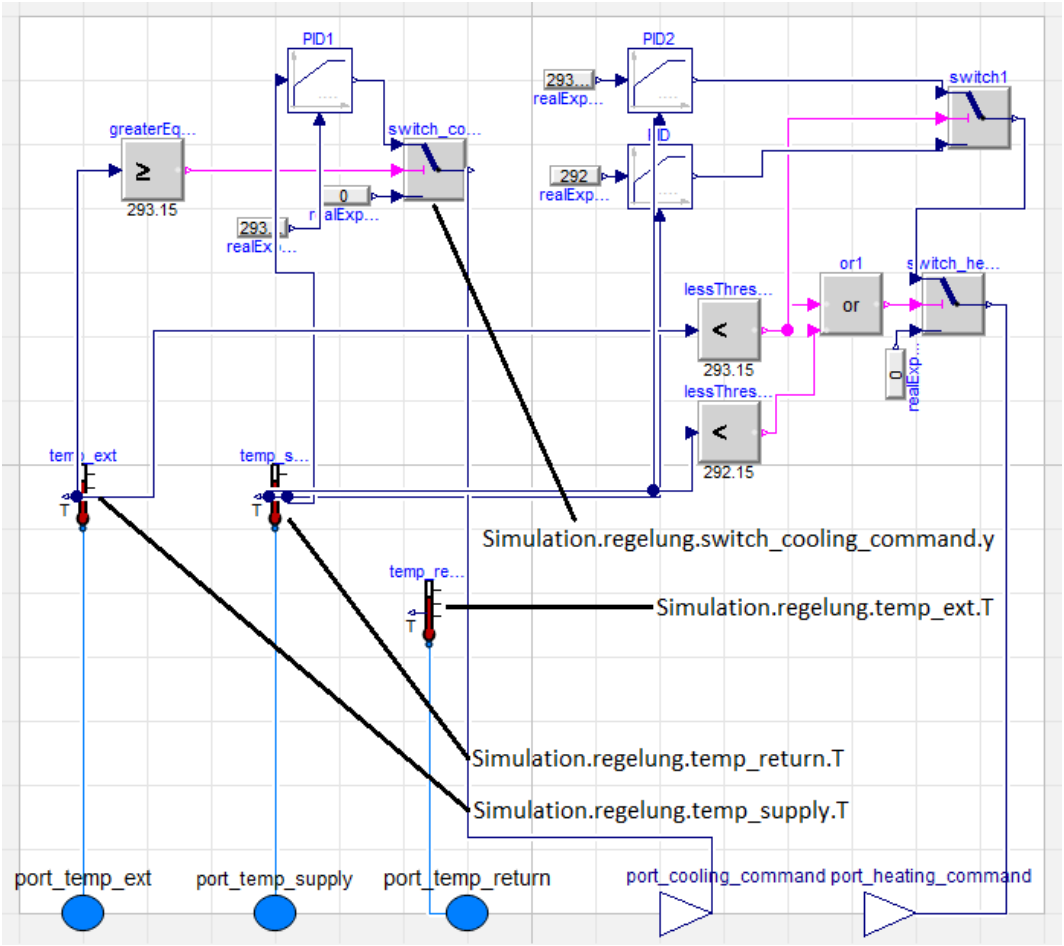


Abb. 7.2: Komponenten in der Regelung die Variablen erzeugen.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Abb. 7.3: Klassifizierung der Vorhersagen bezüglich der Richtigkeit [20].

f1-Score (Formel 7.3): entspricht dem harmonischen Mittelwert von Precision und Recall [24], und wird benutzt da Precision und Recall alleine nicht ausschlaggebend genug sind .

$$f1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{7.3}$$

Accuracy (Formel 7.4): quantifiziert den Anteil an richtigen Vorhersagen, in den gesamten gemachten Vorhersagen.

$$Accuracy = 2 \cdot \frac{TP + TN}{TP + TN + FP + FN} \tag{7.4}$$

Bei einer Klassifikation mit mehreren Klassen, kann die Klassifikation jeder Klasse analog wie in Abbildung 7.3 betrachtet werden; die falschen Ergebnissen verteilen sich unter den anderen Klassen, und die wahren negativen Ergebnisse entsprechen richtigen Vorhersagen bei anderen Klassen.

7.4.2 Ergebnisse mit Wetterdaten aus San Francisco

In Abbildung 7.4 wird ein Bericht zur Bewertung der Leistung des Modells gezeigt. Dort werden die Bewertungskriterien Precision, Recall, und f1-score für jede Klasse die in den Simulationsdaten und auch vom Klassifikator erkannt wurden, berechnet. Die Berechnung für *Accuracy*, umfasst alle Klassen.

	precision	recall	f1-score
Cooling_Valve_Command	0.99	0.98	0.98
Economizer_Damper_Command	0.00	0.00	0.00
Outside_Air_Temperature_Sensor	1.00	0.88	0.94
Return_Air_Temperature_Sensor	0.68	0.76	0.72
Supply_Air_Flow_Sensor	0.00	0.00	0.00
Supply_Air_Temperature_Sensor	0.68	0.67	0.68
accuracy			0.82
macro avg	0.56	0.55	0.55
weighted avg	0.84	0.82	0.83

Abb. 7.4: Bericht des Ergebnisses für die Daten der San Francisco-Simulation.

Abbildung 7.5 zeigt eine Konfusionsmatrix [11], die bei jeder Klasse aufzeigt, welcher Anteil der Elemente der Klasse zu welchen Klassen zugeordnet wurden. Es treten dabei mehr Spalten auf als Brick-Klassen in den Simulationsdaten; da der Klassifikator manche Zeitreihen mit falschen Brick-Klassen klassifiziert hat.

Um einen Vergleich machen zu können, werden in Abbildungen 7.6 und 7.7, dieselben Bewertungen für die Leistung des Klassifikators mit den originalen Testdaten von [24] gezeigt.

Es fällt auf, dass die Leistung des Klassifikators bei den synthetischen Daten schlechter ist als bei den originalen Testdaten von [24]. Dies kann bemerkt werden wenn man in den Berichten, den gewichteten f1-score und die Accuracy überprüft. Der gewichtete f1-score bei den Testdaten liegt bei 0.86, bei den synthetischen Daten aus der San

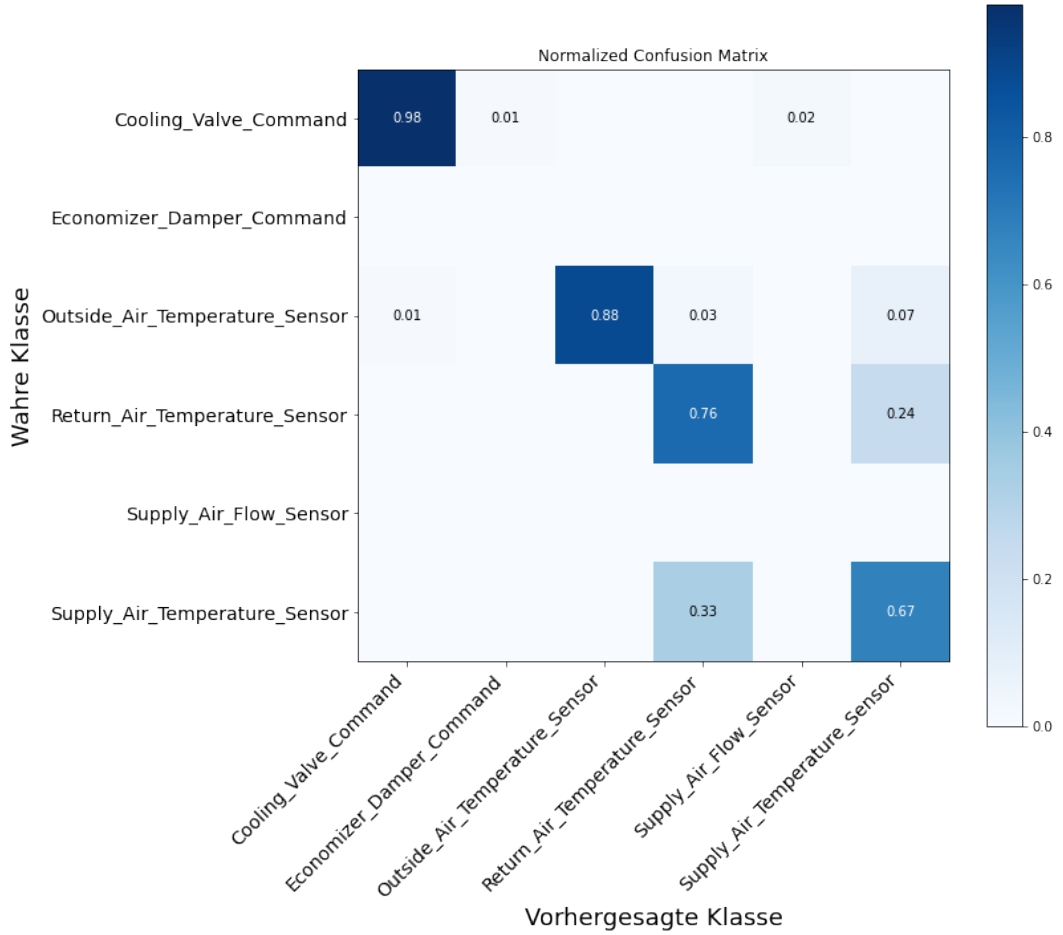


Abb. 7.5: Konfusionsmatrix für die Daten der San Francisco-Simulation.

	precision	recall	f1-score
Cooling_Valve_Command	0.52	0.85	0.65
Damper_Position_Setpoint	0.95	0.76	0.85
Differential_Pressure_Sensor	1.00	1.00	1.00
Economizer_Damper_Command	0.16	0.34	0.22
Outside_Air_Temperature_Sensor	0.84	0.97	0.90
Return_Air_Temperature_Sensor	0.70	0.93	0.80
Supply_Air_Flow_Sensor	0.96	0.85	0.90
Supply_Air_Static_Pressure_Sensor	1.00	0.99	1.00
Supply_Air_Temperature_Sensor	0.89	0.92	0.91
accuracy			0.85
macro avg	0.78	0.85	0.80
weighted avg	0.88	0.85	0.86

Abb. 7.6: Bericht des Ergebnisses für die Testdaten.

Francisco-Simulation liegt der Wert leicht niedriger bei 0.83. Bei der Accuracy passiert etwas ähnliches, 0.85 vs. 0.82.

Die Elemente in der Hauptdiagonale entsprechen den jeweiligen *recall*-Werten. Es ist auch bemerkbar dass bei jeder Klasse die am häufigsten vorhergesagte Klasse auch die richtige ist. Es kann in Abbildung 7.5 auch bemerkt werden, dass der Klassifikator für die synthetischen San-Francisco-Daten die Klassen *Return Temperature Sensor* und

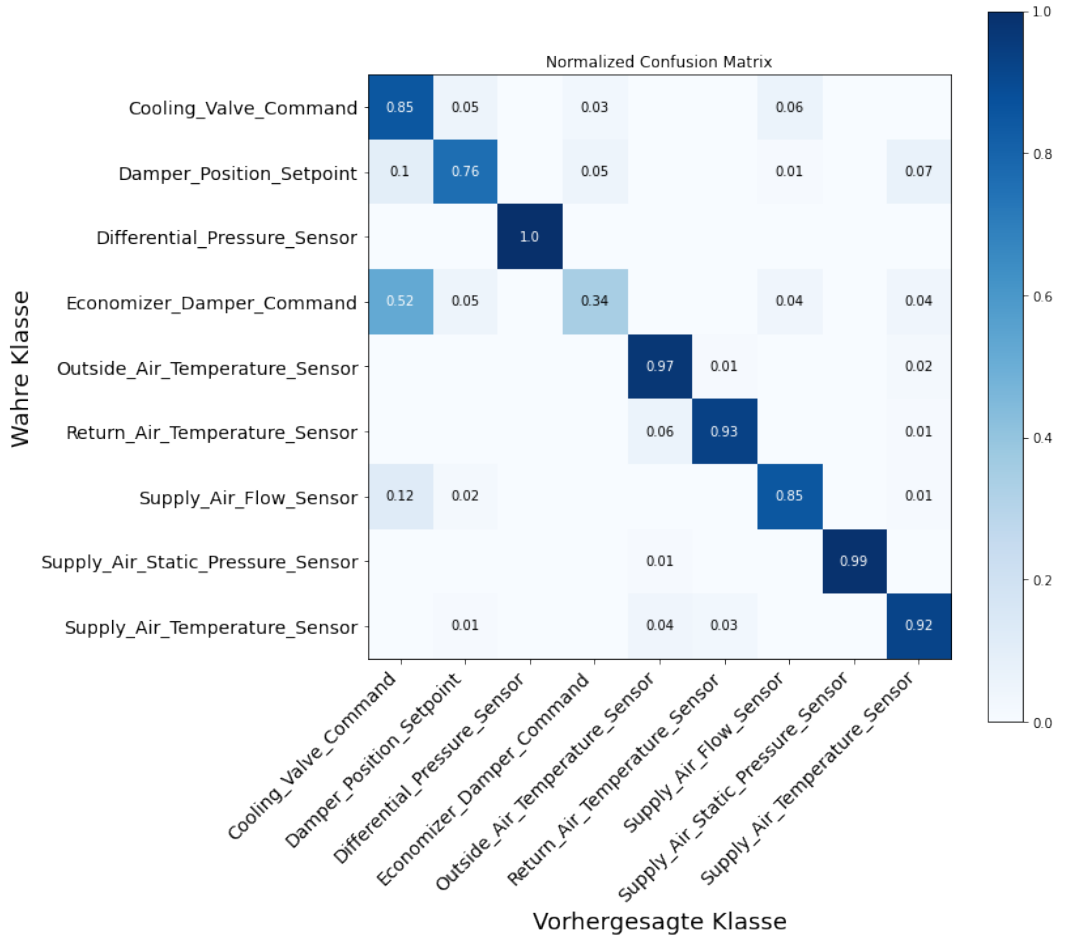


Abb. 7.7: Konfusionsmatrix für die Testdaten des ursprünglichen Modells.

Supply Air Temperature Sensor verwechselt. Sie werden oft miteinander verwechselt und selten als eine von den anderen Klassen zugeordnet. Der Grund dafür soll sein, dass die Verläufe der beiden Klassen von Zeitreihen in den Modelica-Daten qualitativ ähnlich sind, und Wertebereiche haben die nah aneinander liegen (Siehe Abbildung 7.8). Diese Zeitreihen sind auch miteinander verknüpft, da die Klasse *Return Air Temperature Sensor* von *Supply Air Temperature Sensor* abhängig ist.

Bei der Klasse *Cooling Valve Command* ergibt sich einen f1-score von 0.98, ein sehr gutes Ergebnis.

Bei der Klasse *Outside Air Temperature Sensor* ergibt sich ein Recall von 0.88. Es könnte ein besseres Ergebnis erwartet werden, da die benutzten Modelica-Daten reale Außentemperaturmessungen aus San Francisco sind, aus derselben Region woher die Trainingsdaten des ML Modells stammen. Um zu untersuchen warum die Ergebnisse so sind, wird ein Konfidenzband in Abbildung 7.9 erstellt. Das Band umfasst den Bereich zwischen den Kurven des täglichen Verlaufes der Klasse *Outside Air Temperature Sensor* in den Trainingsdaten $\mp$ Standardabweichung. Idealerweise, würde die Kurve der

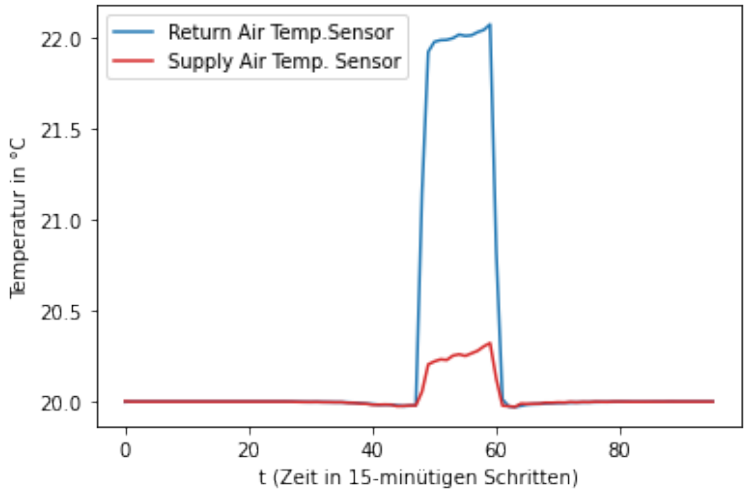


Abb. 7.8: Durchschnittliche tägliche Verläufe der Klassen *Supply Air Temperature Sensor* und *Return Air Temperature Sensor*

Modelica-Daten innerhalb des Konfidenzbandes liegen, in Abbildung 7.9 wird gezeigt dass es hier nicht der Fall ist.

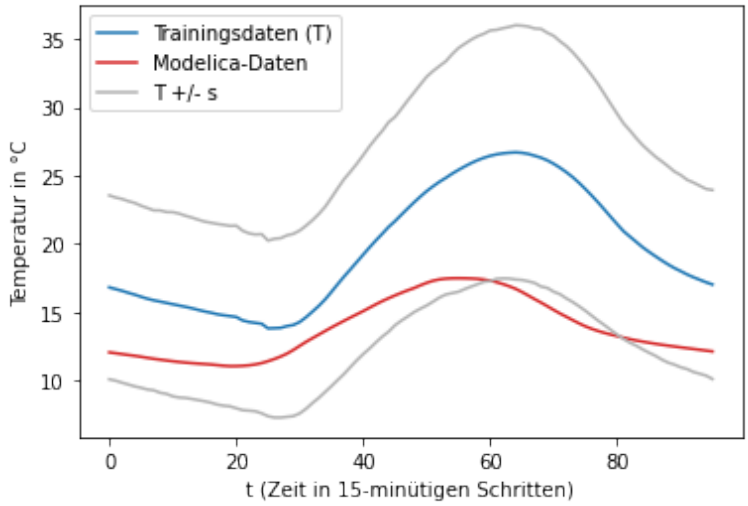


Abb. 7.9: Durchschnittlicher täglicher Verläufe der Zeitreihen der Klasse *Outside Air Temperature Sensor* in den Trainingsdaten und Modelica-Daten mit Konfidenzband.

Vermutlich liegt dies daran, dass in den Trainingsdaten die Anzahl an Zeitreihen im Jahr unregelmäßig verteilt sind. Die Zeitreihen sind im Winter unterrepräsentiert. Dies wird in Abbildung 7.10 gezeigt; es fällt auf, dass es für Dezember gar keine Zeitreihen gibt. Dies hat als Wirkung, dass in den Trainingsdaten die durchschnittliche Zeitreihe wärmere Temperaturen aufweist, als in den Modelica-Daten. Die Daten vom Modelica-Modell sind im Jahr gleichmäßig verteilt.

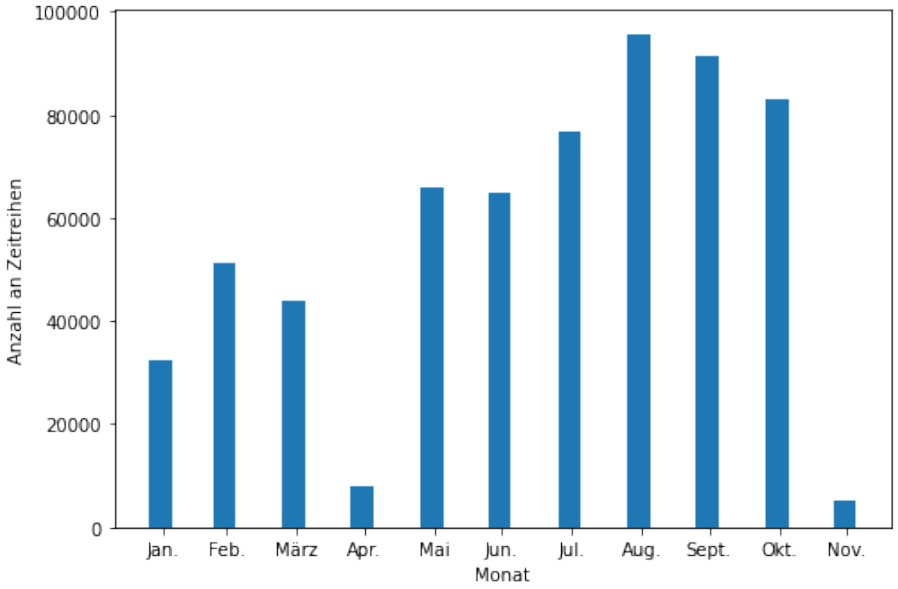


Abb. 7.10: Anzahl an Zeitreihen der Klasse *Outside Air Temperature Sensor* in den Trainingsdaten im Jahr

7.4.3 Ergebnisse mit Wetterdaten aus Chicago

Mit einem f1-score von 0.83, ist das Ergebnis bei der Klasse *Cooling Valve Command* schlechter als in in Abschnitt 7.4.2 (Abbildungen 7.12 und 7.11). Dies liegt daran, dass der Klassifikator viele Zeitreihen der Klasse *Outside Air Temperature Sensor*, als *Cooling Valve Command* eingestuft hat.

	precision	recall	f1-score
Cooling_Valve_Command	0.71	0.99	0.83
Economizer_Damper_Command	0.00	0.00	0.00
Outside_Air_Temperature_Sensor	1.00	0.44	0.61
Return_Air_Temperature_Sensor	0.61	0.82	0.70
Supply_Air_Flow_Sensor	0.00	0.00	0.00
Supply_Air_Temperature_Sensor	0.74	0.60	0.66
accuracy			0.71
macro avg	0.51	0.47	0.47
weighted avg	0.76	0.71	0.70

Abb. 7.11: Bericht des Ergebnisses für die Daten der Chicago-Simulation.

Bei der Klasse *Outside Air Temperature Sensor*, ist das aber deutlich schlechter als in Abschnitt 7.4.2. Der f1-score liegt lediglich bei 0.61. Dies soll daran liegen, dass das ML Modell mit Daten aus Kalifornien trainiert wurde, während die Wetterdaten für diese Modelica-Simulation aus Chicago kommen. Ähnlich wie in Abschnitt 7.4.2 wird ein Konfidenzband in Abbildung 7.13 gezeigt. Die Kurve der Modelica-Daten liegt in diesem Fall mehrheitlich außerhalb des Konfidenzbandes.

Die Klassen *Return Air Temperature Sensor* und *Supply Air Temperature Sensor* wurden hier auch miteinander verwechselt. Der Grund dafür soll derselbe sein wie in Abschnitt

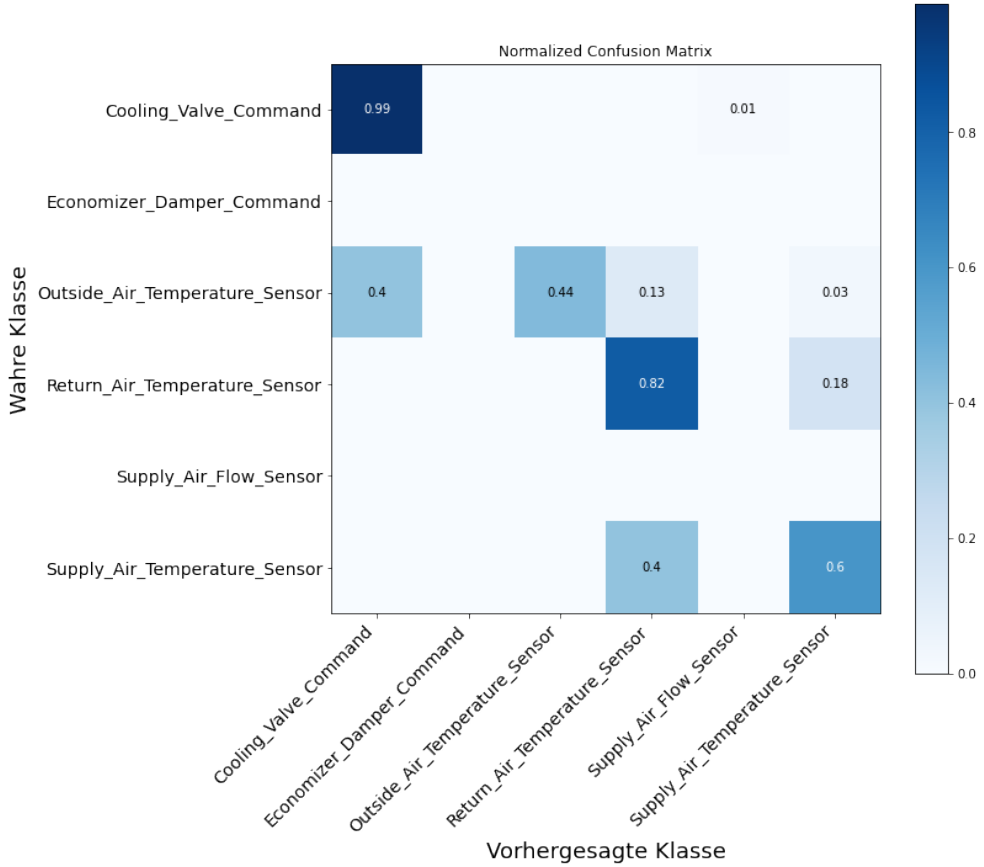


Abb. 7.12: Konfusionsmatrix für die Daten der Chicago-Simulation.

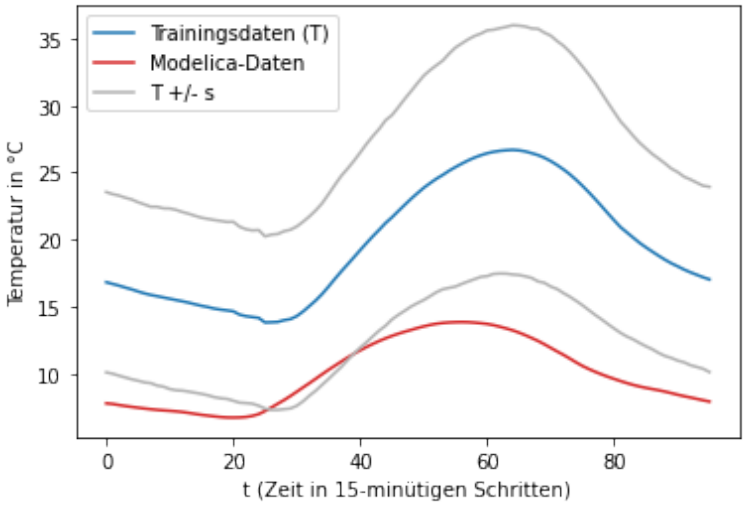


Abb. 7.13: Durchschnittlicher täglicher Verläufe der Zeitreihen der Klasse *Outside Air Temperature Sensor* in den Trainingsdaten und Modelica-Daten mit Konfidenzband (Chicago).

7.4.2. Dennoch sind die f1-scores hier (0.70 und 0.66) schlechter als die analogen in Abschnitt 7.4.2. Die erhöhten falschen Klassifikationen von Zeitreihen der Klasse *Outside Air Temperature Sensor* haben hier zur Verschlechterung der Ergebnisse beigetragen, dies kann in Abbildung 7.12 gesehen werden.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die Anwendung von Anlagensimulationen bei Planungsphasen in der Gebäudetechnik können einen bedeutenden Beitrag zur Energieeffizienz leisten. Softwareanwendungen für die Gebäudeautomation sind in diesem Kontext auch von Nutzen. Daher wäre eine automatisierte Überführbarkeit zwischen Darstellungen gebäudetechnischer Anlagen in Form von semantischen Modellen und Simulationsmodellen wünschenswert. Um diese Möglichkeit zu untersuchen wurden anhand eines Beispielsystems ein semantisches Modell im Brick-Schema und ein Modelica-Simulationsmodell erstellt.

Um das Brick-Modell zu erstellen, wurden in der Brick-Ontologie v1.2 passende Klassen für die originalen Elementen des Anlagenschemas nach Deskription und Funktion gesucht und zugeordnet. Dabei ist aufgefallen, dass es in der Brick-Ontologie an Ausdrucksfähigkeit bei Komponenten die von einem Regelungssystems nicht steuerbar sind, mangelt; es gibt z.B. keine Klassen für Leitungen, Luftauslässe oder Abzweigungen. Der Grund dafür ist, dass das Brick-Schema dafür gedacht ist, die Sensorik und die von der Regelung gesteuerten Komponenten einer Anlage darzustellen. Es wurde auch bemerkt dass bei mehrfachen Verbindungen von Elementen, Mehrdeutigkeiten entstehen können. Bei solchen Fällen wurden im Brick-Modell die Stränge mit Elementen der Klasse *Medium*, und den Brick-Beziehungen *hasInputSubstance* und *hasOutputSubstance* charakterisiert, um die Mehrdeutigkeiten zu beseitigen.

Anhand des Schaltschemas des Beispielssystems, wurde ein Simulationsmodell in der Modellierungssprache Modelica unter Verwendung der Bibliotheken AixLib, Modelica Buildings Library, LibEAS und Modelica Standard Library erzeugt. Ähnlich wie bei Brick, wurden in den benutzten Bibliotheken keine entsprechenden Komponentenmodelle für mehrere Komponenten wie z.B. Lüftungshauben und Luftauslässe gefunden; sie wurden durch *Buildings.Fluid.FixedResistances.PressureDrop* Modelle repräsentiert. Es wurden vereinfacht eine Regelung entworfen und Wärme- und Feuchteeinträge dargestellt; damit können Rahmenbedingungen die den des Beispielsystems ähnlich sind simuliert werden. Als Parameter wurden vereinfacht für die Heiz- und Kühlregister Werte aus Beispielen der LibEAS, und für weitere Komponenten aus Standardwerten der Modelica Buildings Library, ausgewählt; der Nennvolumenstrom entspricht dem originalen des Beispielsystems.

Um die Möglichkeit der Durchführung von automatisierten Überführungen zu untersuchen, wurden Tabellen mit den verschiedenen Klassen von Komponenten in den Darstellungen (Schaltschema, Brick und Modelica; jeweils in eine eigene Spalte) erstellt und verglichen. Es wurde angenommen, dass die Schaltschemata maschinenlesbar sind;

daher wurden sie in der Untersuchung auf Überführbarkeit auch mitbetrachtet. Eine vollständige und eindeutige automatisierte Überführung zwischen den Darstellungen in den Spalten würde Surjektivität oder Bijektivität benötigen. Bei den vorhandenen Darstellungen wäre diese Bedingung für die Beziehungen zwischen den Spalten nicht erfüllt; und damit eine automatisierte Überführung schwierig.

Mit der Nutzung von Wetterdaten aus der Modelica Buildings Library und dem erstellten Modelica-Modell wurden zwei Jahressimulationen erstellt. Dies wurde gemacht um ein Klassifikationsmodell von Mertens [24] der Methode*ExtraTrees* des machinellen Lernens auf die resultierenden synthetischen Messdaten anzuwenden und dabei die Leistung dieses Modells zu bewerten. Die zwei benutzten Wetterdateien kommen aus den USA; aus Chicago und San Francisco. Die resultierenden synthetischen Messdaten wurden umformatiert, um sie dem optimalen Format der Eingabedaten des Klassifikationsmodells anzupassen. Dabei wurden die Simulationsdaten in Zeitreihen von 24 Stunden aufgeteilt. Es wurde ein Resampling durchgeführt, und die Abtastperiode der Daten von 1 Minute zu 15 Minuten geändert. Als Plausibilitätskontrolle wurden die Ergebnisse der Variablen *Simulation.regelung.temp_supply.T* und *Simulation.regelung.temp_return.T* überprüft.

Unter den synthetischen Messdaten wurden 4 Variablen gefunden, welche Brick-Klassen entsprechen, und vom Klassifikationsmodell erkannt werden können. Diese sind: *Cooling Valve Command*, *Outside Air Temperature Sensor*, *Return Air Temperature Sensor* und *Supply Air Temperature Sensor*. Bei den Messdaten die mit der Wetterdatei aus San Francisco simuliert wurden, ist das Ergebnis bei der Klasse *Cooling Valve Command* sehr gut, mit einem f1-score von 0.98. Die Klassen *Outside Air Temperature Sensor* und *Return Air Temperature Sensor* wurden vom Klassifikator verwechselt, da deren Verläufe im Wertebereich und auch qualitativ ähnlich sind. Bei der Klasse *Outside Air Temperature Sensor* ist der f1-score 0.90; es könnte trotzdem ein besseres Ergebnis erwartet werden, da es sich von realen Messdaten aus derselben Region der Trainingsdaten handelt. Dieses Ergebnis erklärt sich, da die Verteilung der Zeitreihen der relevanten Klasse in den Trainingsdaten über das Jahr unregelmäßig ist. Bei der Chicago-Simulation sind die Ergebnisse schlechter, da die Trainingsdaten des Klassifikationsmodells nicht aus derselben Region kommen. Die Klasse *Outside Air Temperature Sensor* wurde dabei schlechter klassifiziert, mit einem f1-score von 0.61; sie wurde oft mit der Klasse *Cooling Valve Command* verwechselt, was ihren f1-score auch verschlechtert (0.83). Die Klassen *Outside Air Temperature Sensor* und *Return Air Temperature Sensor* wurden vom Klassifikator auch oft miteinander verwechselt.

8.2 Ausblick

Es können Ansätze verwendet werden um eine automatisierte Überführung zu ermöglichen, z.B. die Nutzung der Brick-Klasse *Tags*, oder eine gezielte Wahl der Brick-Klassen. Um jedoch eine automatisierte Überführung zu ermöglichen müssten die Darstellungen speziell erstellt werden. Dies würde implizieren, dass bei Darstellungen die schon vorhanden sind und nicht mit diesem Ziel erstellt wurden, keine automatisierte Überführung möglich wäre. Die Anwendung von Ansätzen für dieses Problem könnte dazu führen, dass sich differenzierte Praktiken etablieren; dies könnte die Standardisierung des Brick-Schemas schädigen.

Deswegen sollten Vorgehensweisen und Änderungen die das Ziel haben, automatisierte Überführungen zu ermöglichen, in der Brick-Ontologie berücksichtigt werden. Vorschläge für Brick können direkt in GitHub, und auch in der Brick-Ontology Arbeitsgruppe gemacht werden. Dies wäre auch für Vorgehensweisen um die Verbindungen zwischen Elementen in Brick besser zu charakterisieren wichtig. Auf diese Weise kann die Standardisierung der Brick-Ontologie erhalten bleiben. Eine Erweiterung der Brick-Klassen, besonders für nicht-steuerbaren Komponenten wie z.B. Leitungen, Luftauslässe oder Abzweigungen wäre in diesem Kontext auch von Vorteil.

Andere Metadaten-Schemata wie die TUBES-Ontologie, oder die Brick+ Ontologie (eine Erweiterung der Brick-Ontologie) könnten in Hinblick auf automatische Überführbarkeit zusammen mit Modelica untersucht werden.

Bei der Bewertung des Klassifikators, haben sich bei Zeitreihen von manchen Brick-Klassen gute Ergebnisse bewiesen. Es kann vermutet werden, dass der umgekehrte Vorgang, d.h. die Klassifikation von realen Daten mit einem Klassifikator der mit synthetischen Daten trainiert wurde; erfolgreich sein wird. Dennoch waren die Klassifikationen von Zeitreihen anderer Klassen nicht optimal. Es hat sich auch gezeigt, dass die Wetterbedingungen der Trainingsdaten und der klassifizierten Daten einen großen Einfluss auf die Ergebnisse haben. Dies soll bei weiteren Klassifikationsaufgaben von Datenpunkten betrachtet werden.

Literatur

[1] Gaudenz Alder. Flowchart Maker and Online Diagram Software. `https://draw.io/`, 2020. Accessed: 2021-05-18.

[2] Modelica Association. Modelica standard library. `https://github.com/modelica/ModelicaStandardLibrary`, 2020.

[3] Bharathan Balaji, Arka Bhattacharya, Gabriel Fierro, Jingkun Gao, Joshua Gluck, Dezhi Hong, Aslak Johansen, Jason Koh, Joern Ploennigs, Yuvraj Agarwal, Mario Berges, David Culler, Rajesh Gupta, Mikkel Baun Kjærgaard, Mani Srivastava, and Kamin Whitehouse. Brick: Towards a Unified Metadata Schema For Buildings. In *Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments*, BuildSys '16, page 41–50, New York, NY, USA, 2016. Association for Computing Machinery.

[4] Arka Bhattacharya, Joern Ploennigs, and David Culler. Short Paper: Analyzing Metadata Schemas for Buildings:The Good, the Bad, and the Ugly. In *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, BuildSys '15, page 33–34, New York, NY, USA, 2015. Association for Computing Machinery.

[5] Carl Boettiger. rdflib: A high level wrapper around the redland package for common rdf applications, 2018.

[6] CalcWorkshop. What is a Bijection. `https://calcworkshop.com/functions/bijection/`, February 2021. Accessed: 2021-05-18.

[7] Climate-Data.org. Climate Data for Cities Worldwide. `https://en.climate-data.org/`, 2021. Accessed: 2021-05-10.

[8] The Brick Consortium. Brick Studio. `https://brickschema.github.io/brick-studio/`. Accessed: 2021-05-18.

[9] The Brick Consortium. Brick Viewer. `https://viewer.brickschema.org/`. Accessed: 2021-05-18.

[10] The Brick Consortium. Brick. A uniform metadata schema for buildings. `https://brickschema.org/`, 2021. Accessed: 2021-05-18.

[11] David Cournapeau. Confusion Matrix. `https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html`, 2020. Accessed: 2021-05-10.

[12] Microsoft Docs. XML-Serialisierung. `https://docs.microsoft.com/de-de/dotnet/standard/serialization/introducing-xml-serialization`, 2017. Accessed: 2021-05-18.

[13] Hilding Elmqvist and Sven Erik Mattsson. An introduction to the physical modeling language Modelica. In *Proceedings of the 9th European simulation symposium, ESS*, volume 97, pages 19–23. Citeseer, 1997.

[14] Michael Wetter et al. Modelica Buildings Library. *LBNL Report*, 2015.

[15] Gabe Fierro, Marco Pritoni, Moustafa Abdelbaky, Daniel Lengyel, John Leyden, Anand Prakash, Pranav Gupta, Paul Raftery, Therese Peffer, Greg Thomson, and David E. Culler. Mortar: An Open Testbed for Portable Building Analytics. *ACM Trans. Sen. Netw.*, 16(1), December 2019.

[16] Bundesministerium für Wirtschaft und Energie (BMWi). Energieeffizienz in Zahlen 2020. 2020.

[17] Fraunhofer IIS. LibEAS Modelica Library. Unveröffentlicht.

[18] Klemm Ingenieure. Fraunhofer-Institut IIS in Dresden. Nutzungsspezifische Anlagen. Raumlufttechnik. Unveröffentlicht.

[19] C. Maria Keet. The African wildlife ontology tutorial ontologies. *Journal of Biomedical Semantics*, 11(1):4, 2020.

[20] Will Koehrsen. Beyond Accuracy: Precision and Recall. *Towards Data Science*, 2018.

[21] Jason Koh. Brick Tutorial at BuildSys 2017. `https://github.com/BuildSysUniformMetadata/brick-tutorial-buildsys2017`, 2017. Accessed: 2021-05-18.

[22] Miguel Manotas. Modelling and calculation of the heat and mass transfer in the hydraulic system of a liquid-cooled data center. Unveröffentlichte Studienarbeit, 2020.

[23] Joep Meindertsma. What's the best RDF serialization format? `https://ontola.io/blog/rdf-serialization-formats/`, June 2019. Accessed: 2021-05-18.

[24] Noah Mertens. Automatisierte Klassifikation von Datenpunkten raumlufttechnischer Anlagen anhand von Merkmalen der Zeitserien. Unveröffentlichte Diplomarbeit, 2020.

[25] Dirk Mueller, Moritz Lauster, Ana Constantin, Marcus Fuchs, and Peter Remmen. AixLib – An Open-Source Modelica Library within the IEA-EBC Annex 60 Framework. 09 2016.

[26] MS Nikulin and M Hazewinkel. Encyclopedia of mathematics. *URL: http://www. encyclopediaof-math. org/index. php*, 2001.

[27] The Institute of Specialist Surveyors and Engineers. Specific average moisture generation rates. `https://www.isse.org.uk/articles/dampness`. Accessed: 2021-05-18.

[28] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[29] Linda R Petzold. Description of dassl: a differential/algebraic system solver. Technical report, Sandia National Labs., Livermore, CA (USA), 1982.

[30] Alessandro Picarelli. Dymola and modelica explained. `https://www.claytex.com/tech-blog/dymola-and-modelica-explained/`, 2019. Accessed: 2021-05-18.

[31] Jaak Simm, Ildefons Magrans de Abril, and Masashi Sugiyama. Tree-Based Ensemble Multi-Task Learning Method for Classification and Regression. *IEICE TRANSACTIONS on Information and Systems*, 97(6):1677–1681, 2014.

[32] Matthew Stevens. Human Body Heat as a Source for Thermoelectric Energy Generation. `http://large.stanford.edu/courses/2016/ph240/stevens1/`. Accessed: 2021-05-18.

[33] Florian Stinner, Yingying Yang, Thomas Schreiber, Gerrit Bode, Marc Baranski, and Dirk Mueller. Generating Generic Data Sets for Machine Learning Applications in Building Services Using Standardized Time Series Data. 05 2019.

[34] Ritika Tiwari. Regression vs Classification in Machine Learning: What is The Difference? `https://in.springboard.com/blog/regression-vs-classification-in-machine-learning/`, June 2020. Accessed: 2021-05-18.

[35] Mingjun Wei. Automatisierte Klassifikation und Bewertung von Betriebszuständen raumlufttechnischer Anlagen. Unveröffentlichte Diplomarbeit, 2019.

Anhang

A Ontologie des Brick-Modells

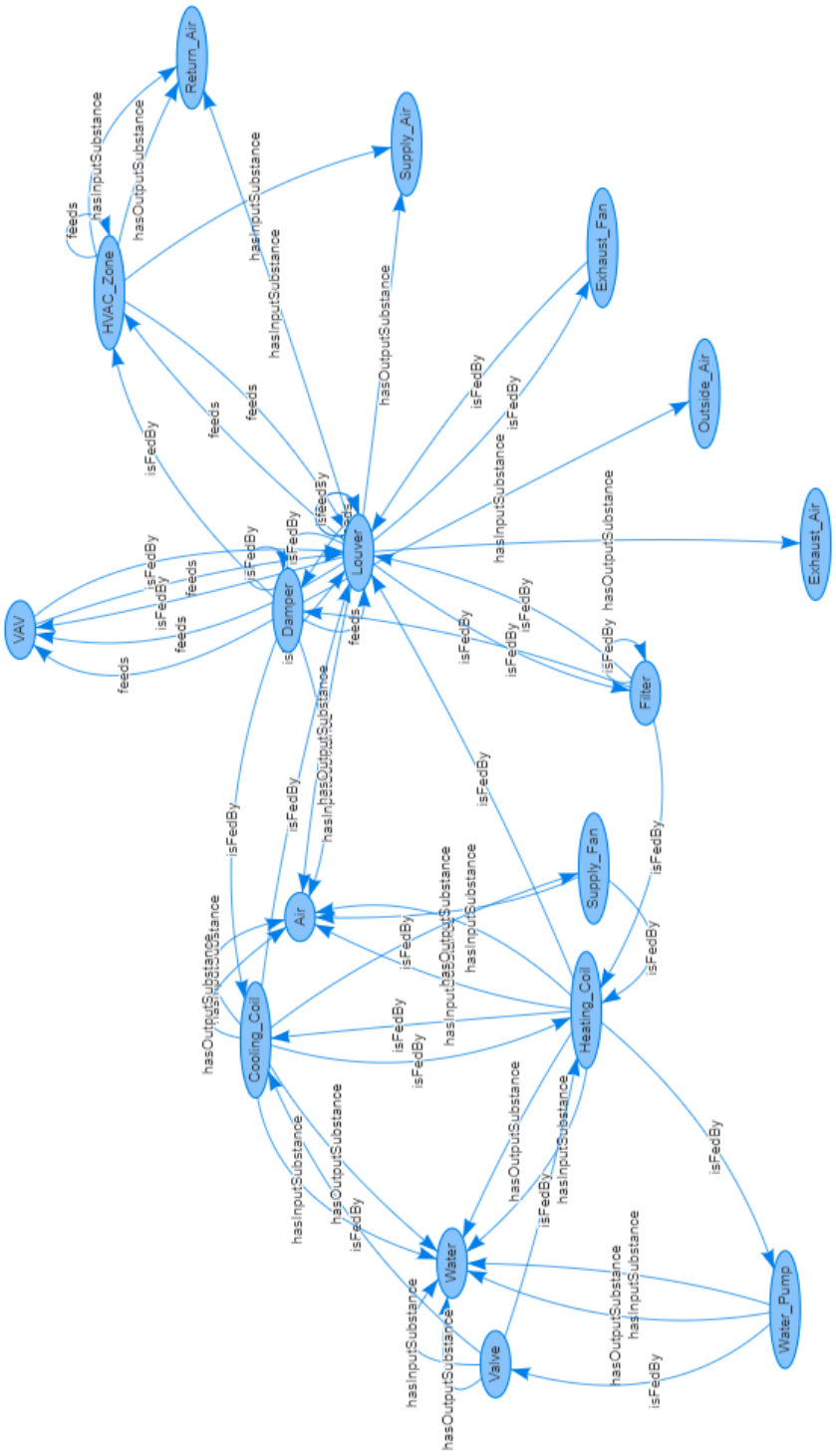


Abb. A.1: Ontologie des in dieser Arbeit erstellten Brick-Modelles

B Detailliertere Darstellung von Teilbereichen der RLT-Anlage in Brick

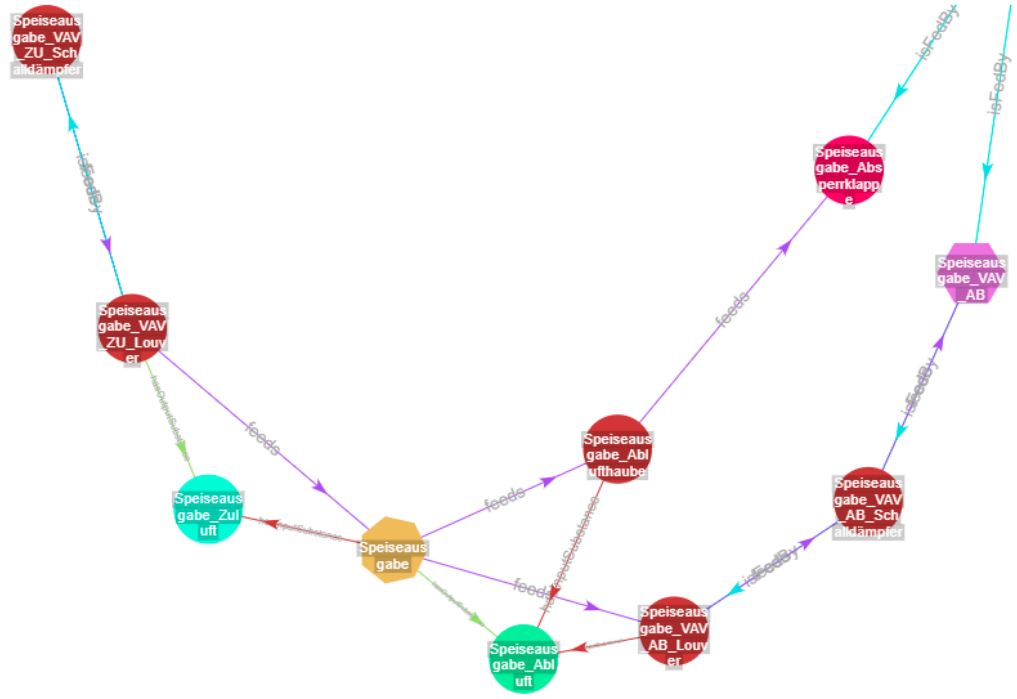


Abb. B.1: Speiseausgabe

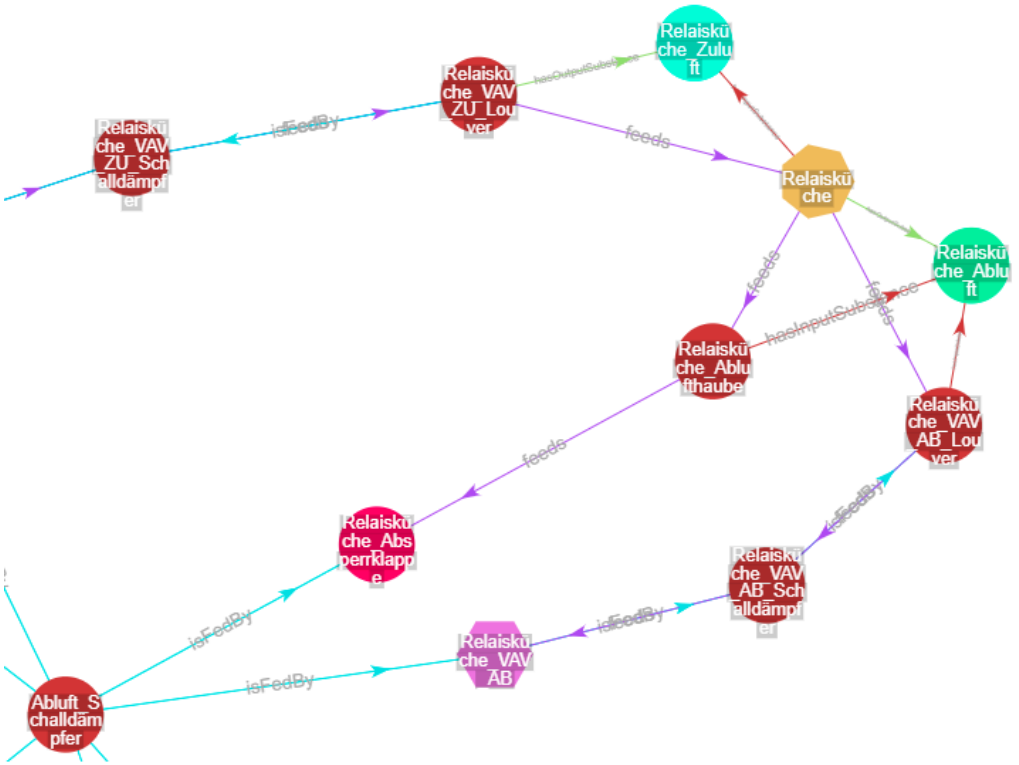


Abb. B.2: Relaiskueche

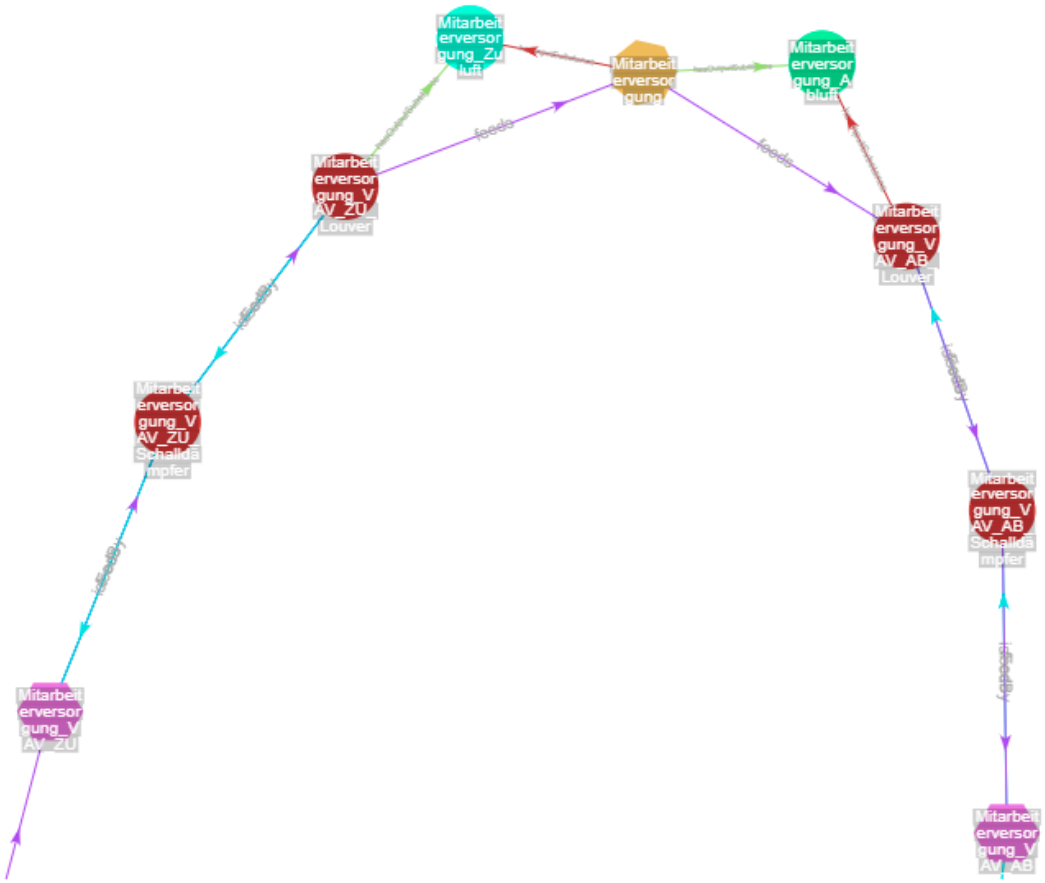


Abb. B.3: Mitarbeiterversorgung

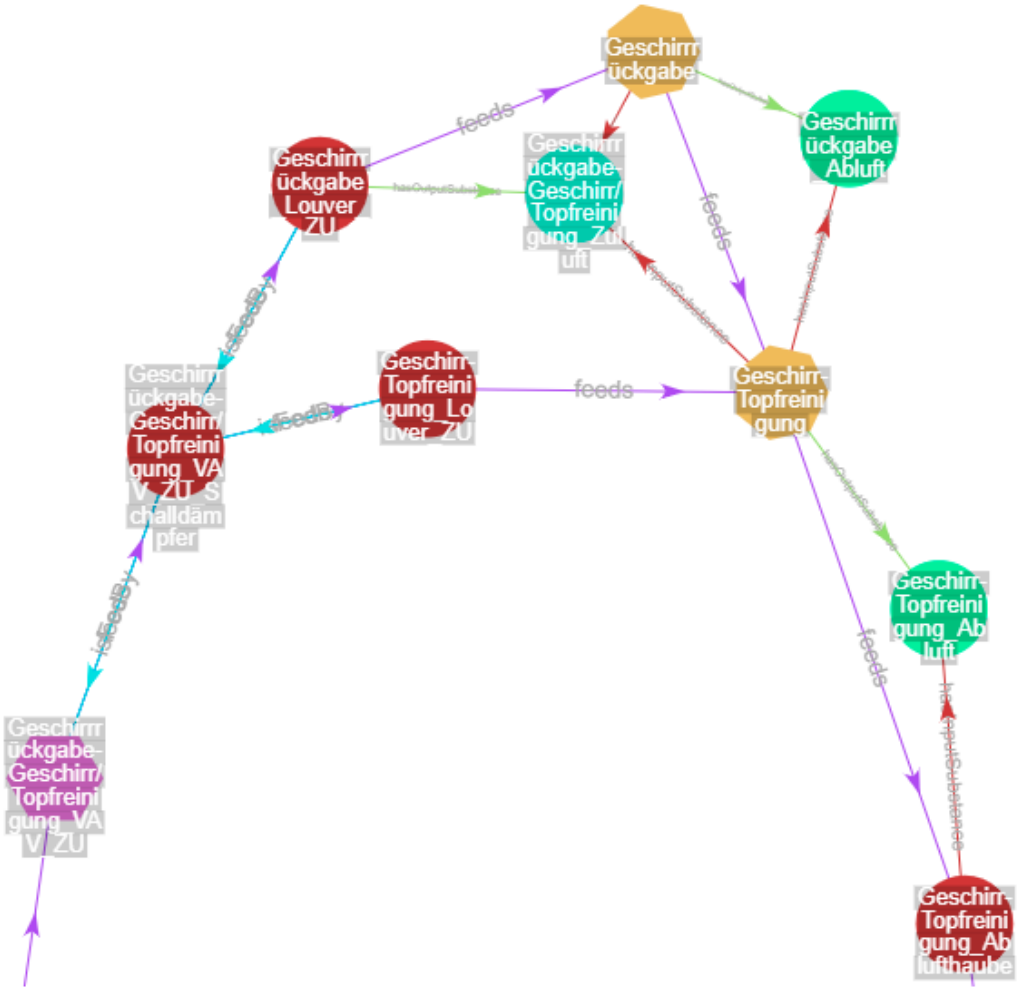


Abb. B.4: Geschirr- und Topfreinigung

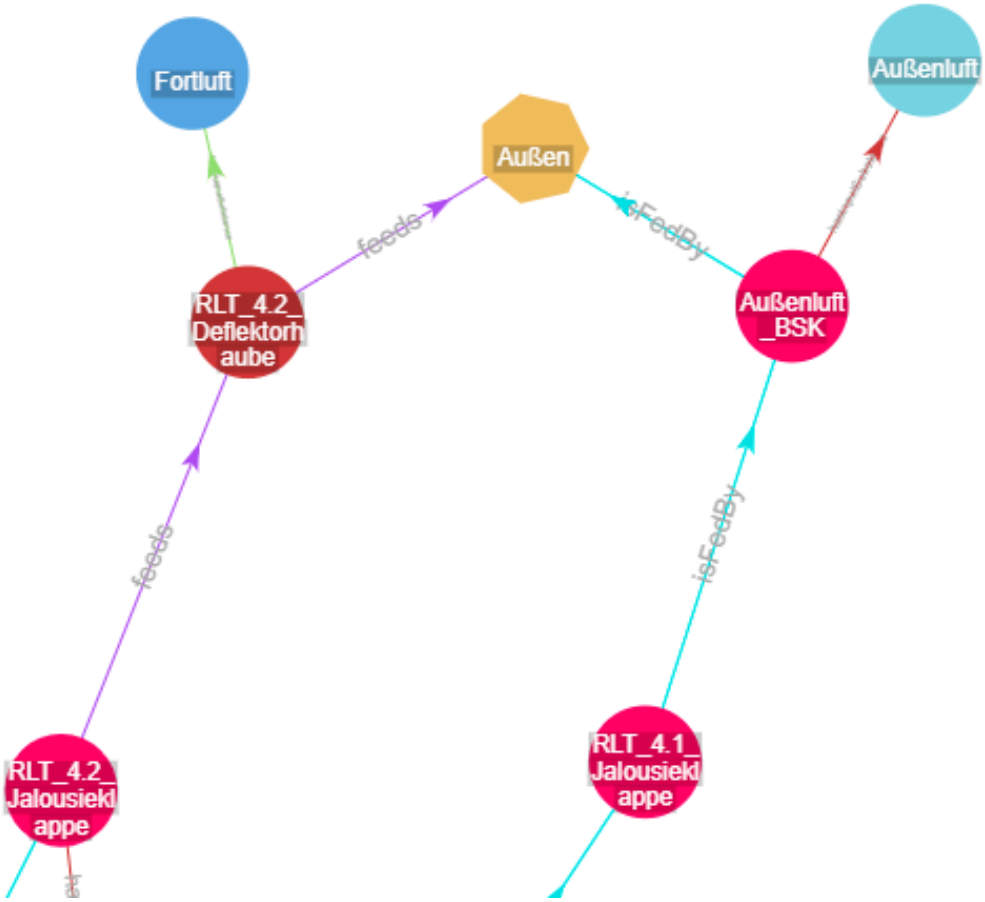
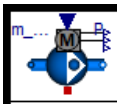
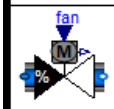
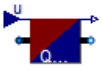


Abb. B.5: Verbindung der RLT-Anlage mit der Umgebung

C Zeichenerklärung zu den Symbolen der Modelica-Komponenten

	Buildings.Fluid.Movers.FlowControlled_m_flow
	Buildings.Fluid.Actuators.Valves.TwoWayEqualPercentage
	Buildings.Fluid.Actuators.Valves.TwoWayQuickOpening
	Buildings.Fluid.FixedResistances.PressureDrop
	Buildings.Fluid.HeatExchangers.WetCoilCounterFlow
	LibEAS.RLT.Bauteile.HeatRecovery
	Buildings.Fluid.HeatExchangers.HeaterCooler_u
	Buildings.Fluid.MixingVolumes.MixingVolume
	AixLib.Fluid.FixedResistances.Junction

Eidesstattliche Erklärung

Hiermit versichere ich, *Miguel Manotas*, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von mir angegebenen Quellen angefertigt zu haben. Alle aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche gekennzeichnet. Die Arbeit wurde noch keiner Prüfungsbehörde in gleicher oder ähnlicher Form vorgelegt.

Dresden, den 30.06.2021

_____

Unterschrift mit Vor- und Nachname