Technische Universität Dresden

Fakultät Elektrotechnik und Informationstechnik

Institut für Regelungs- und Steuerungstheorie

Studienarbeit

Semantische Katalogisierung und formale Repräsentation regelungstechnischer Systemmodelle

vorgelegt von: Jonathan Rockstroh
geboren am: 14. Mai 1997 in Pirna

Betreuer:	Dr.-Ing. C. Knoll
Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack
Tag der Einreichung:	11. August 2021

Bitte ersetzen Sie diese Seite vor dem Binden mit der Aufgabenstellung.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage an der Fakultät Elektrotechnik und Informationstechnik eingereichte Studienarbeit zum Thema

Semantische Katalogisierung und formale Repräsentation regelungstechnischer Systemmodelle

selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, wurden als solche kenntlich gemacht.

Pirna, 1. September 2021 Jonathan Rockstroh

Kurzfassung

An dieser Stelle fügen Sie bitte eine deutsche Kurzfassung ein.

Abstract

Please insert the English abstract here.

Inhaltsverzeichnis

Verzeichnis der Formelzeichen

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Inhalt: Kurze Erklärung warum ein Katalog von Modellen sinnvoll ist und was die Idee attraktiv macht.

1.2 Präzisierung der Aufgabenstellung

Inhalt: Aufgabenstellung in stichpunktartigen Sätzen.

Im Rahmen dieser Studienarbeit soll eine Katalog für regelungstechnische Systeme entworfen werden. Darin sollen Modelle als Textrepräsentation und (optional) zusätzlich als implementierter Code enthalten sein. Für beide Repräsentationsarten soll es eine einheitliche Repräsentationsweise geben. Die Umsetzung so erfolgen, das neue Modelle möglichst einfach hinzugefügt werden können. Ebenso soll ein Klassifikationssystem erstellt werden mit dem die Modelle innerhalb der regelungstechnischen Theorie eingeordnet werden können. Das Klassifikationssystem soll auf eine signifikante Anzahl regelungstechnischer Veröffentlichungen angewandt werden. Außerdem sollen ausgewählte Modelle implementiert werden.

Original (letzter Part): Ziel der Arbeit ist es, mittels sogenannter ontologischer Methoden ein Klassifikationsystem zu erstellen und auf eine signifikante Anzahl (z.B. 50) regelungstechnischer Veröffentlichungen anzuwenden. Zudem sollen die wichtigsten Modelle aus den Veröffentlichungen in Python implementiert und mittels einer Hierarchie semantischer Eigenschaften (z.B. "nichtlinear", SZustandsdimension: 8", "Flachheitsstatus: nicht flach") erfasst werden.

Kapitel 2

Vorbetrachtung

Der in 3 vorgestellte Katalog und dem dazugehörigen Klassifikationssystem ist das Resultat vieler Entscheidungen. Dem ist eine Recherche von Literatur mit regelungstechnischen Modellen voraus gegangen. Das Kapitel beginnt mit einer Erklärung zu den Begriffen *System* und *Modell*. Anschließend wird in 2.2 der aktuelle Stand der Modellrecherche und vorhandener Modellsammlungen vorgestellt. Danach werden in 2.3 die Anforderungen für den Katalog formuliert und getroffene Entscheidungen vorgestellt, die zur Erfüllung der Anforderungen beitragen sollen. Bevor in 2.5 die wichtigsten Entscheidungen zur Erstellung des Klassifikationssystems vorgestellt werden, gibt es in 2.4 noch eine Einführung zu Wissensrepräsentationen und die Graphentheorie.

2.1 Dynamische Systeme und Modelle

Die Begriffe *System* und *Modell* kommen in der regelungstechnischen Literatur häufig vor. Die Bedeutung kann, je nachdem in welchen Anwendungsgebiet diese verwendet werden, stark variieren. Und selbst innerhalb regelungstechnischer Literatur werden beide Begriffe eher selten extra definiert oder referenziert, sondern auf Basis eines intuitiven Verständnisses verwendet. Für diese Arbeit, die den Begriff *Systemmodelle* schon im Titel trägt, sind beide Begriffe bedeutsam, weshalb an dieser Stelle beschrieben wird wie diese Begriffe in dieser Arbeit verstanden werden sollen.

Ein *System* ist nach der Norm *DIN IEC 60050-351:2014-09*[1] eine „Menge miteinander in Beziehung stehender Elemente, die in einem bestimmten Zusammenhang als Ganzes gesehen und als von ihrer Umgebung abgegrenzt betrachtet werden“.
Des weiteren wird in den Anmerkungen der Norm ausgeführt, das die Elemente eines Systems „natürliche oder künstliche Gegenstände sowie Arten von Denkvorgängen und deren Ergebnisse“ sein können und dass das System „von der Umgebung und von den anderen äußeren Systemen durch eine gedachte Hüllfläche abgegrenzt betrachtet“ wird.

Ein System wird in dieser Arbeit als *reales System* bezeichnet, wenn dessen Element

[1] siehe [4] Seite 21

nachweisbare Bestandteile der Realität sind.

Modelle sind (nach Stachowiak[20] S. 131ff) Abbildungen oder Repräsentationen natürlicher oder künstlicher Originale, welche selbst wieder Modelle sein können(*Abbildungsmerkmal*). Zudem haben Modelle immer nur eine Auswahl der Attribute des Originals, welche dem Modellersteller und/oder dem Modellnutzer als relevant erscheinen(*Verkürzungsmerkmal*) und Modelle dienen einem bestimmten Zweck(*pragmatisches Merkmal*).

Ein System oder Modell ist *dynamisch*, wenn sich dessen Werte mit der Zeit verändern. Die Regelungstechnik beschäftigt sich unter anderem damit das zeitveränderliche Verhalten von Systemen und Modellen zu untersuchen und zu beeinflussen. Deshalb sind für diese Arbeit nur dynamische Systeme und Modelle von Interesse und die Begriffe *System* und *Modell* implizieren in dieser Arbeit die Eigenschaft *dynamisch*.

Modelle von Systemen(*Systemmodelle*) bestehen in der Regelungstechnik zum einen aus einer graphischen Repräsentation des Systems, die wiederum aus definierten Elementen besteht welche reale Entitäten des Systems repräsentieren und zum anderen aus einem mathematischen Modell, welches das Verhalten des Systemmodells beschreibt. Zu einem System kann es verschiedene Modelle in Form einer graphischen Repräsentation geben, zu denen es wiederum verschiedene mathematische Modelle geben kann(vgl. [12] Sektion 2.1).
Ein *mathematisches Modell* ist ein Modell das die Anwendung mathematischer Methoden erlaubt([6], S.9). Das mathematische Modell ist in der Regelungstechnik normalerweise ein System von Gleichungen.

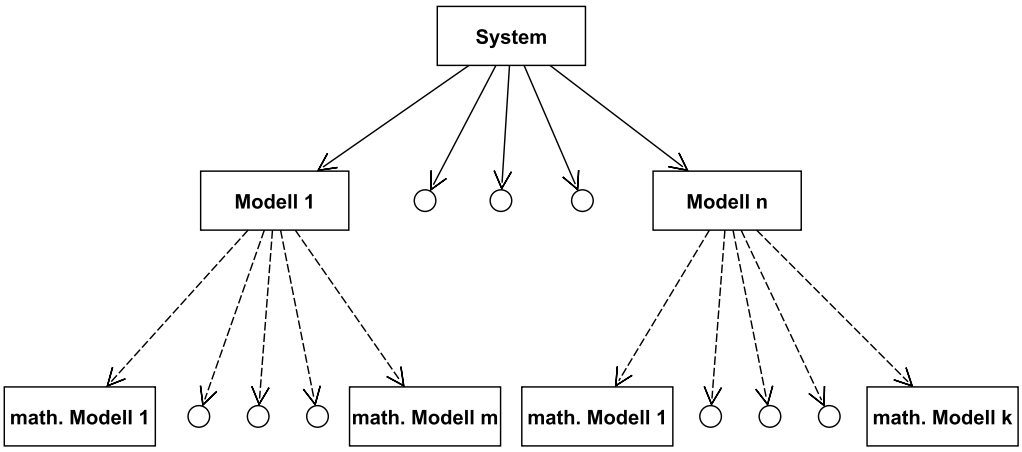

Abbildung 1 – Zusammenhang Systeme, Modelle und mathematische Modelle

Zudem gibt es in der Regelungstechnik Modelle, die ausschließlich aus dem mathematischen Modell, also einer Beschreibung des dynamischen Verhaltens, bestehen. Diese Modelle repräsentieren kein Original und werden in dieser Arbeit als *abstrakte Modelle* bezeichnet. Abstrakte Modelle lassen sich in den obigen Modellbegriff integrieren, indem

gesagt wird das für jedes abstrakte Modell ein System gedacht werden kann, welches nicht bekannt ist.

Der Begriff *System* bezeichnet im Folgenden dynamische Systeme. Der Begriff *Modell* bezeichnet im Folgenden sowohl dynamische Systemmodelle als auch dynamische abstrakte Modelle.

2.2 Aktueller Stand

Für die Zusammenstellung von Wissen wird eine Wissensbasis benötigt. Um diese zu erlangen und um geeignete Modelle für den Katalog zu finden erfolgte eine Modellsuche mit folgenden Erkenntnissen.

Aktuelle Situation der Modellfindung:

1. Regelungstechnische Modelle finden sich aktuell meist verteilt in wissenschaftlichen Publikationen, wie z.B. Lehrbüchern, Artikeln, Dissertationen, Diplom- und Studienarbeiten.

2. Die Qualität der Modelldarstellung ist uneinheitlich. Das die Modellgleichungen eindeutig gekennzeichneten und gemeinsam notiert, sowie die eingeführten Variablen gut beschrieben und klar definierten Typs (Parameter, Eingangs-, Zustandsvariable) sind ist nicht immer gegeben.

    - Beispiel 1: In [11] wird auf Seite 135 das Modell eines dynamischen Systems übersichtlich dargestellt. Es ist abhängig von den Parametern σ, r und b. Die Zustandsvariablen und der Parameter σ werden direkt nach den Modellgleichungen beschrieben. Die Parameter r und b haben hingegen keinen Namen und werden nur als Gleichungen repräsentiert. Der Parameter b hängt wiederum von a ab, welches in dem Artikel auch nicht explizit eingeführt wird.

    - Beispiel 2: In [23] werden die Variablen am Anfang alle eingeführt. Das Modell wird ausführlich hergeleitet. Eine zusammengestellte Übersicht der Modellgleichungen fehlt jedoch. Die Zustandsvariablen müssen aus Ausgangsvektor und Abbildungen erschlossen werden. Die Modellgleichungen sind im Artikel verteilt.

3. Die mathematische Darstellungsform der Modellgleichungen kann sich unterscheiden.

    - Beispiel 3: In [19] Seite 14 werden die Modellgleichungen als Gleichungssystem von Differentialgleichungen erster Ordnung dargestellt. Allerdings mit zusätzlichen Summanden auf der linken Seite der Gleichung.

- Beispiel 4: In [2] Seite 3 wird die Modellgleichung als Differentialgleichung zweiter Ordnung dargestellt.

- Beispiel 5: In [8] Seite 168f, Beispiel B.3 werden die Modellgleichungen als Gleichungssystem von Differentialgleichungen zweiter Ordnung dargestellt, wobei die linke Seite der Gleichung aus Summanden und Produkten besteht.

4. Die Modelleigenschaften sind oft nur implizit gegeben, z.B. kann bei einem Steuerungsentwurf geschlussfolgert werden, dass das untersuchte System stabil ist. Die explizite Nennung von Modelleigenschaften erfolgt meist nur, wenn diese für die Publikation von Relevanz sind.

   - Beispiel 6: Im Artikel [14] Seite 761, letzter Abschnitt wird auf die Steuerbarkeit der Modelldarstellung eingegangen. Andere Eigenschaften finden keine Erwähnung.

5. In nahezu allen Publikationen erfolgt die Erprobung der Ergebnisse mittels Simulation.

   - Beispiel 7: In [2] wurde der Eingang in die Modellgleichung eingesetzt. Für die Implementation musste dieser wieder extrahiert werden. Die Eingangsgröße ist nicht die Kraft, welche normalerweise für mechanische Systeme zu erwarten ist, sondern die Auslenkung. Für eine Darstellung mit der Kraft als Eingang wäre eine weitere Umformung nötig.

   - Beispiel 8: In [5] Seite 10910, Fig. 8 werden die Eingangswerte als grauer Graph dargestellt. Eine Darstellung als Gleichung fehlt. Ebenso fehlt bei den verwendeten Parameterwerte zum Beispiel der Wert für die Gleichspannung v_{DC}.

6. Die genutzte Implementation wird nicht publiziert bzw. veröffentlicht.

Die beschriebenen Sachverhalte in den Beispielen sind nicht zwangsläufig als Kritik gemeint. Es kann gute Gründe dafür geben. Für die Erfassung der Situation sind diese aber nicht von Bedeutung. Eine Beleuchtung möglicher Gründe findet deshalb nicht statt.

Feststellung:
Die zielgerichtete Suche nach Modellen, z. B. mit bestimmten Eigenschaften, ist oft eine zeitintensive und aufwendige Angelegenheit. Zudem braucht es häufig zusätzliche Eigenarbeit um zu einer brauchbaren Modelldarstellung zu gelangen. Die Implementierung muss aktuell fast immer von eigener Hand erfolgen. Für die Validierung des eigenen Codes und die Reproduktion der Resultate einer Publikation ist eine softwaretechnische Implementation des Modells sowie der daran angehängten Umgebung (Steuerung, Regelung, Beobachter etc.) oft notwendig (vgl. [9], Seite 1). Durch obige Aspekte ist das meist aufwendig oder nicht möglich.

Aktuelle Situation von Modellsammlungen und -katalogen:
Bestehende Zusammenstellungen von regelungstechnischen Modellen sind schwer zu finden. Das kann daran liegen, das wenige existieren. Es könnte aber auch daran liegen, das diese einfach nur schwer zu finden sind. Zum Beispiel, weil diese von Suchmaschinen als irrelevant eingestuft werden und folglich sehr weit hinten in den Suchergebnissen landen. Zudem basieren die Suchergebnisse nur auf den eingegebenen Wörtern. Ein Verständnis für den Kontext fehlt. Eine sehr große Anzahl von Suchergebnissen ist die Folge. Es ist also durchaus plausibel das mehr als die beiden Zusammenstellungen, die im Folgenden kurz vorgestellt werden, existieren.

The Automatic Control Knowledge Repository (ACKRep):
Das in [7] vorgestellte ACKRep[2] ist ein Tool, mit dem Ziel die implementierten Ergebnisse von Publikationen reproduzierbarer zu machen. Der Teil der *ProblemSpecification* ist eine Sammlung von Modellen, die als Python-Quellcode repräsentiert werden. Weitere Informationen zu den Modellen sind in einer YAML-Datei hinterlegt.

Beispiele unteraktuierter mechanischer Systeme in [10]:
Im Zuge der Betrachtung unteraktuierter mechanischer Systeme werden in dem Artikel die Modellgleichungen von 11 Beispielen tabellarisch in einer vereinheitlichten Form aufgeführt. Es wurden Differentialgleichungen zweiter Ordnung als Darstellungsform gewählt. Zudem werden die Systeme bezüglich ihrer Beschränkungen, ihrer Konfigurationscharakteristik und ihrer regelungstechnischen Problemstellung klassifiziert.

Im ACKRep liegt der Fokus auf der Implementierung der Modelle. Ein weiterer Fokus liegt auf der Auffindbarkeit der Modelle innerhalb des ACKReps. Dafür ist eine Suchfunktion angedacht, die auf die Informationen aus der YAML-Datei zugreift. Die in [9] eingeführte *Ontology of Control Systems Engineering (OCSE)* soll zu einer einheitlichen Benennung der Modellinformationen beitragen.
In [10] liegt der Fokus auf der menschlichen Lesbarkeit. Die Modellgleichungen sind als Text bzw. Formeln repräsentiert und ermöglichen die Anwendung von analytischen Methoden.

Beide Zusammenstellungen verwenden eine einheitliche Form für die Modellrepräsentation. Zudem findet bei beiden eine Klassifikation statt, welche sich allerdings in Umfang und Inhalt unterscheiden.

2.3 Anforderungen an den Katalog

Anforderungen an den Katalog:
Der Katalog soll den Prozess der Modellfindung und Nutzung vereinfachen, sodass die in Abschnitt:„Aktueller Stand“ beschriebenen Schwierigkeiten nicht durchlaufen werden

[2] ACKRep GitHub Repository: https://github.com/ackrep-org/ackrep_data

müssen. Daher wurden folgende Anforderungen an den Katalog gestellt:

Anforderung A.1: Neue Modelle sollen einfach und unkompliziert zu finden sein.

Anforderung A.2: Die Modelleigenschaften sollen so gut wie möglich erfasst sein. Das heißt:

- Sie sollen möglichst vollständig sein.
- Sie sollen in einer übersichtlichen Darstellung aufgelistet sein.
- Sie sollen einer einheitlichen Namensgebung folgen.
- Sie sollen eine klare Definition haben.

Anforderung A.3: Die Modelle sollen eine einheitliche Darstellungsform haben.

Anforderung A.4: Die Variablen, deren Typ und Bedeutung sollen in einer sinnvollen, einheitlichen Darstellung notiert sein.

Anforderung A.5: Die Modelle sollen möglichst implementiert vorliegen. Die Implementierung soll einfach verwendbar sein.

Anforderung A.6: Der Katalog soll erweiterbar sein.

Die Anforderung **A.6** macht es notwendig den Fall einer großen Anzahl von im Katalog existierenden Modellen mitzudenken. Mit zunehmender Modellanzahl wird es komplizierter bestimmte Modelle innerhalb der Ordnerstruktur des Kataloges zu finden. Außerdem sollte auch die Auffindbarkeit von Modellen anhand bestimmter Modelleigenschaften beachtet werden. Das macht eine Suchfunktion erstrebenswert um die Anforderung **A.1** zu erfüllen. Die Erstellung einer Suchfunktion soll durch folgende Entscheidung erleichtert werden:

Entscheidung E.1: Zu jedem Modell soll eine Datei *(Metadaten-Datei)* geben, in der wichtige Informationen wie der Modellschlüssel und -Name, die Modelleigenschaften und der Modellersteller hinterlegt werden. Die Metadaten-Datei soll im einfach les- und editierbaren YAML Format vorliegen.

Die Idee und Umsetzung von Entscheidung **E.1** basiert auf [7]. Die Struktur der Metadaten-Datei wurde aus dem *ACKRep* übernommen und leicht angepasst.

Anforderung **A.2** soll durch das in der Aufgabenstellung geforderte *Klassifikationssystems (KS)* und die Anwendung dessen auf die Modelle erfüllt werden. Die Auflistung der Modelleigenschaften erfolgt in der Metadaten-Datei. Die Namen der Attribute im KS stellen eine einheitliche Namensgebung sicher. Die Definition der Attribute und die Relationen zwischen diesen basieren auf im KS enthaltenen Referenzen.

Entscheidung E.2: Die Einträge des KS, welche unter anderem Namen, Relationen zu anderen Einträgen und Wertetyp enthalten werden im YAML Format gespeichert. Um eine grafische Darstellung des KS zu erhalten soll ein Python-Skript geschrieben werden.

Wie vollständig die in der Metadaten-Datei enthaltenen Modelleigenschaften sind hängt davon ab, wie gut erforscht das Modell ist und wie viel Zeit in das Anlegen des Modells gesteckt wird.

In Textform dargestellt Modelle haben den Vorteil das Elemente wie Tiefstellung grafisch dargestellt werden können. Im Fließtext muss die Umsetzung der Elemente über ein Notationssytem erreicht werden, welches die Lesbarkeit verringert. Die Notation der Modellgleichungen in einer für Menschen gut lesbaren Form ist erstrebenswert, damit die Modelle für die Anwendung von analytischen Methoden einfacher verwendbar sind.

Entscheidung E.3: Die Modellgleichungen und die zugehörigen Variablen werden in Textform notiert. Grundlage ist ein LaTeX-Dokument, das in eine PDF-Datei umgewandelt wird. Die Modellgleichungen werden als System von Differentialgleichungen erster Ordnung dargestellt. Die Auflistung der Variablen und ihrer Bedeutung erfolgt gebündelt.

Die Bündelung der Variablendefinitionen innerhalb der Textrepräsentation macht diese gut im Dokument auffindbar. Die Verwendung von DGLn erster Ordnung ist zum einen Allgemein, da DGLn höherer Ordnung durch Einführung definitorischer Gleichungen in ein System von DGLn erster Ordnung überführt werden können, und zum anderen die in Publikationen am häufigsten verwendete Form zur Darstellung der Modellgleichungen.

Entscheidung E.4: Die Modelle werden in der Programmiersprache Python implementiert. Dabei soll jedes Modell als Python-Klasse umgesetzt werden.

Die Implementierung der Modelle als Klasse soll den zweiten Teil von Anforderung **A.5** erfüllen. Die Modelle werden in der Verwendung als Objekt instanziiert, was für das Verständnis eine einfache und intuitive Parallele zur realen Welt ist in der Systeme als physische Objekte existieren. Zudem können innerhalb des instanziierten Objektes eine Reihe von Informationen zu dem spezifisch implementierten Modell gespeichert und abgefragt werden. Die Verwendung für eigene Simulationen ist damit sehr unkompliziert, da die innerhalb der Klasse enthaltenen Modellgleichungen über die Methoden der Klasse direkt verwendet werden können.

Es stellte sich noch die Frage, ob alle Modelle des Kataloges implementiert vorliegen müssen. Nun liegt zum einen eine formale und einheitliche Modelldarstellung schon allein durch die Textform vor. Zum anderen sind einige Modelle recht schwer zu implementieren, wie z. B. Modelle deren Verhalten durch partiellen Differentialgleichungen beschrieben werden. Da auch solche Modelle mit in den Katalog aufgenommen werden sollen, wurde

die folgende Entscheidung getroffen.

Entscheidung E.5: Die Implementierung eines Modells ist optional.

Um die Verwendbarkeit der vorliegenden Modelle weiter zu vereinfachen erschien es sinnvoll ein Beispielset der Parameterwerte zur Verfügung zu stellen. Die Beispielwerte sollen in beiden Notationsformen vorliegen.

Entscheidung E.6: Zu jedem Modell soll es ein Beispielset für die Parameterwerte geben. Diese sollen sowohl in der Textform als auch in der implementierten Form Anwendung finden. Die Beispielwerte sollen redundanzfrei notiert werden.

Die Entscheidungen **E.3** und **E.4** haben als Folge, das für eine Erweiterung des Kataloges eine Einarbeitung in die Form der Implementation und der Textrepräsentation erfolgen muss. Die folgende Entscheidung soll die Einarbeitung für neue Nutzer aber auch generell das Erstellen von neuen Modellen vereinfachen. Außerdem soll diese zur konstanten Erfüllung der Anforderungen **A.3** und **A.4** bei Erweiterung des Kataloges durch neue Personen beitragen.

Entscheidung E.7: Für Implementation und Textrepräsentation sollen Vorlagen erstellt werden, welche die repetitiven Elemente, wie z.B. die Struktur, dieser enthält.

Für die Anforderungen **A.1** und **A.6** ist es wichtig einen einfachen Zugang zu den Daten des Kataloges zu schaffen, denn zum einen werden für die individuelle Nutzung der Modelle die Dokumentation und Implementation als lokale Daten benötigt und zum anderen werden für das Anlegen neuer Modelle die Vorlagen lokal vorliegen.

Entscheidung E.8: Der Katalog als ein Git-Repositorium angelegt werden.

Git ist eine dezentrale Versionsverwaltungssoftware, die sich für die Arbeit an textbasierten Dateistrukturen bewährt hat. Ein Git-Repositorium(*Git-Repo*) ist eine Menge von versionsverwalteter Informationen und steht üblicherweise für ein Dateisystem bestehend aus Ordnern und Dateien. Ein Git-Repo kann als lokale Kopie heruntergeladen und lokal editiert werden. Die Änderungen einer lokal bearbeiteten Kopie eines Git-Repos können auf das originale oder offizielle Git-Repo übertragen werden. Das geschieht in Form einer Anfrage zur Übertragung (*Pull-Request*), welche durch die Inhaber des offiziellen Repos geprüft wird. Wird eine Pull-Request angenommen, so werden die im lokalen Repo vorgenommenen Änderungen auf in das offizielle Repo übertragen(vgl. [7] Abschnitt 2.1).

Die folgenden Abschnitte behandeln den Prozess um das Wissen über Modelle und deren Modelleigenschaften formal darzustellen.

2.4 Wissensrepräsentationen

Wissensrepräsentationen sollen das existierende Wissen innerhalb eines Wissensbereichs formal abbilden. Für die Bestandteile der Repräsentation wird ein explizites Vokabular genutzt und die Beziehungen zwischen diesen werden definiert (vgl. [1] S.60, [18] S.1). Eine häufig verwendete Möglichkeit um eine formale Darstellung von Wissensrepräsentationen zu erhalten ist die Verwendung von Methoden und Begriffen aus der Graphentheorie.

Exkurs: *Graphentheorie*[3]
Ein *Graph* besteht aus einer Menge von *Knoten* N und *Kanten* E. Die Kanten verbinden die Knoten. Die Menge E besteht also aus 2-Element Untermengen von N. Ist der Graph *gerichtet*, so hat der Graph die zwei Funktionen $s : E \to N$ und $t : E \to N$ die jeder Kante e einen Ursprungsknoten s(e) und einen Endknoten t(e) zuweisen. Kann der Graph mehrere Kanten zwischen denselben zwei Knoten haben, dann wird dieser *gerichteter Multi-Graph* genannt. Zusätzlich kann die Menge L und die Funktion l eingeführt werden. Die Menge L umfasst die Namen der Knoten und Kanten. Die Funktion $l : N \cup E \to L$ weißt jedem Knoten n und jeder Kante e einen Namen aus der Menge l zu. Ein gerichteter *Pfad* ist ein Graph $P = (N, E)$, der zwei Knoten x_1 und x_i eines Graphen G über eine Sequenz von Kanten $(e_1, \ldots, e_i)$ verbindet. Ein Knoten kann maximal ein Mal entlang eines Pfades vorkommen. Zwei erwähnenswerte Spezialfälle von Graphen sind Bäume und kreisfreie Graphen. Ein Baum liegt vor, wenn jeder Knoten n ϵ N genau einen Nachfolger besitzen.„Ein Graph heißt kreisfrei, wenn es keinen Knoten gibt der einen Pfad zu sich selbst hat. Kreisfreie gerichtete Graphen werden auch DAG(engl. **D**irected **A**cyclic **G**raph) genannt“[4].

In Wissensrepräsentationen, die sich der Graphentheorie bedienen, steht jeder Knoten für einen Bestandteil der Repräsentation und jede Kante stellt eine Beziehung zwischen zwei Bestandteilen dar. Der Name der Kante definiert die Art der Beziehung. Für die Wissensrepräsentation gibt es verschiedene Formen die sich in ihrer semantischen Vielfalt unterscheiden. Beispiele solcher Formen sind: Taxonomie, Thesaurus, Semantisches Netz. Auf die Taxonomie und das Semantische Netz wird im Folgenden kurz eingegangen.

Die *Taxonomie* ist eine der einfachsten Formen der Wissensrepräsentationen. Die Bestandteile dieser sind Kategorien die in eine Hierarchie aus Ober- und Unterkategorien gesetzt werden. Jede Kante stellt eine „ist auch “ Beziehung dar. Jede Unterkategorie ist auch ein Element ihrer Oberkategorie. Die Darstellung kann über einen DAG erfolgen. *Semantische Netze* „sind Graphen die Begriffe und ihre Relationen zueinander darstellen“[5]. Im Unterschied zur Taxonomie sind die Namen der Kanten nicht auf eine Bezeichnung festgelegt. Durch diese verschiedenen Kantentypen werden die unterschiedlichen Beziehungen zwischen den Bestandteilen des Netzes beschrieben. Das Semantische Netz weist zudem verschiedene Knotentypen auf. Knoten können Kategorien sein, die

[3] siehe [21] S.29 und [3] Abschnitte 1.1, 1.3 und 1.10
[4] 21.
[5] [21] S.28

sich wie in der Taxonomie in Ober- und Unterkategorien aufteilen. Die Kanten zwischen Knoten, die Kategorien darstellen, sind mit „ist auch“ bezeichnet. Knoten in Semantischen Netzen können zudem Werte und konkrete Objekte bestimmter Kategorien repräsentieren. Zudem spezifiziert das Netz bestimmte charakteristische Eigenschaften, die alle Elemente eines Knotens gemeinsam haben. Diese charakteristischen Eigenschaften werden in einem Semantischen Netz durch Beziehungen zu anderen Knoten des Netzes dargestellt. Semantische Netze haben häufig eine ‚Open-World‘Annahme. Wenn in einem Semantischen Netz eine Aussage nicht enthalten ist wird nicht automatisch davon ausgegangen, das diese nicht gilt.

Der Begriff der *Ontologie* wird im Zuge von Wissensrepräsentationen häufig verwendet. Nach Studer [22] ist eine Ontologie im Sinne der Informatik eine „formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung“. Der Begriff *formal* bedeutet zum Beispiel, dass die Ontologie maschinenlesbar sein soll. Eine Diskussion zu den Schwächen dieser Definition führt Stuckenschmidt in [21] im Abschnitt *1.4 Anmerkungen zur Begrifflichkeit*.

2.5 Erstellung des Klassifikationssystems

Das *Klassifikationssystem (KS)* soll eine Wissensrepräsentation zu den Eigenschaften von regelungstechnischen Modellen darstellen. Die Frage: „Welchen Umfang soll das KS haben?“ ist dabei eine zentrale Frage gewesen, denn Sie entscheidet welche Wissensbereiche das KS abdecken soll. Die folgende Entscheidung beantwortet diese Frage und basiert auf der Überlegung, welches Wissen nötig ist um eine einheitliche Benennung der Modelleigenschaften zu erreichen.

Entscheidung E.9: Das Klassifikationssystem soll nur den Teilbereich des Wissens der Mathematik, sowie Regelungs- und Steuerungstheorie enthalten, der sich auf regelungstechnische Systeme und Modelle bezieht.

Eine Folgefrage zu der Entscheidung **E.9** war: Sollen selten verwendete und eher unbekannte Eigenschaften mit in das KS genommen werden? Dafür spricht zum einen, das auch eher unbekannte Eigenschaften zu dem in Entscheidung **E.9** benannten Wissensbereich dazugehören und das es für die weitere, zukünftige Ausarbeitung des KS einfacher ist einmal hinzugefügte Eigenschaften wieder zu entfernen, als diese neu zu finden. Dagegen spricht, dass das KS mit der Intention erstellt wird aktiv verwendet zu werden. Die Komplexität des KS niedrig zu halten ist dafür sinnvoll. Eine Entscheidung zu dieser Frage wurde bisher nicht getroffen.

Des weiteren galt es zu Entscheiden, welche Form der Wissensrepräsentation das KS haben soll. Die dahinter stehende Frage lautet: Welche semantischen Elemente werden benötigt bzw. sollen verwendet werden, um das darzustellende Wissen zu repräsentieren? Die Antworten zu dieser Frage wurden im Prozess der Erstellung gegeben. Die folgenden

Entscheidung fasst diese zusammen.

Entscheidung E.10: Das KS soll ein Semantisches Netz sein.

Die Namensgebung ist ein zentrales Element im KS.

Entscheidung E.11: Die Namen der Eigenschaften im KS sind eindeutig. Es gibt eine definierte Menge von Kantennamen, welche die Arten der Beziehung zwischen zwei Knoten definieren.

Für dieselbe Eigenschaft können in seltenen Fällen verschiedene Begriffe in der Literatur auftauchen. In einem solchen Fall soll der in der Praxis geläufigere Begriff verwendet werden. Ein Beispiel für einen solchen Fall sind die Begriffe *Zeitvarianz* und *Zeitvariabilität*(s. [13], S. 114[6]). Beide Bezeichnen die Eigenschaft eines Modells, das die Modellgleichungen explizit von der Zeit abhängig sind. Perspektivisch kann für solche Fälle, das Element *synonym* in die Menge der Kantennamen aufgenommen werden.
Folgend werden zwei Problemstellungen vorgestellt, die ursächlich für die zwei nächsten Entscheidungen waren.

Problemstellung: **Linearität**
Die Begriffe *Linearität*, *Linear* und *Nichtlinear* werden in der Regelungstechnik häufig verwendet, weshalb diese Begriffe auch im KS auftreten sollen. Da *Linear* und *Nichtlinear* spezifische Formen der *Linearität* sind wurden diese Begriffe als Werte der Kategorie *Linearität* im KS eingeführt. Die Werte *Linear* und *Nichtlinear* müssen aber auch als Kategorien im KS existieren, weil es weitere Eigenschaften gibt, die nur auf lineare oder nichtlineare Systeme zutreffen. Dazu wurde folgende Entscheidung getroffen.

Entscheidung E.12: Alle Kategorien haben Wertetypen, die festlegen welche Art von Werten diese bei Anwendung des KS auf ein Modell haben können. Werteknoten können Kategorien sein.

Das ein Knoten mehr als einen Typ haben kann stellt eine Abweichung in der Formulierung von Semantischen Netzen dar. Diese wurde in das KS eingeführt, weil es für die Abbildung des geforderten Wissens und für den gewünschten Praxisbezug des Klassifikationssystems als sinnvoll erachtet wurde.
Der Knoten zur Nichtlinearität wurde mit *Strictly_Non_Linear* bezeichnet, weil die Bezeichnung *Non_Linear* oder *Nonlinear*, bei der ausschließlichen Betrachtung des Begriffs, auch so interpretiert werden kann, dass ein mit dieser Eigenschaft versehenes Modell nicht zwangsweise Linear ist. So interpretiert, könnte ein Modell mit dieser Eigenschaft auch Linear sein. Um diese Interpretation zu vermeiden wurde die Bezeichnung *Strictly_Non_Linear* gewählt, welche die Eigenschaft des nicht linearen betonen soll.

[6] Im selben Abschnitt der *Zeitvariablen Systeme* wird auch die Bezeichnung *zeitvaraint* verwendet.

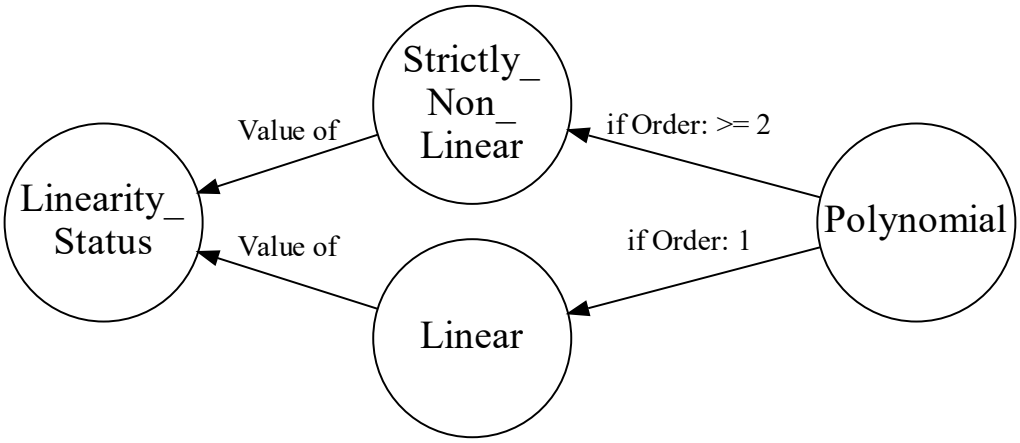

Abbildung 2 – Verworfene Integrationsart für die Kategorie *Polynom* im KS

Problemstellung: **Polynom**
Ein Polynom kann abhängig von seiner Ordnung linear oder nichtlinear sein. Um diesen Sachverhalt vollständig im KS abzubilden müsste das Polynom im KS ein Objekt der Kategorien *Linear* und *Strictly_Non_Linear* sein, welche zwei Werte der Kategorie *Linearity* sind, die sich gegenseitig ausschließen. In der Anwendung auf ein Modell lässt sich dieser Widerspruch im Allgemeinen problemlos aufheben, da bei einem spezifischen Modell, dessen Modellgleichungen eine polynomiale Form haben, die Ordnung der Polynome festgelegt ist. Die durch diese Problemstellung aufgetretene Frage ist also: Sollen Widersprüche im KS auftreten dürfen, wenn diese sich bei Anwendung auf konkrete Modelle aufheben?

Entscheidung E.13: Innerhalb KS soll es keine Widersprüche geben.

Der Grund für die Entscheidung ist, dass es in Anbetracht der Anwendbarkeit für sinnvoller erachtet wurde das KS widerspruchsfrei zu gestalten. In Folge dieser Entscheidung wurde das Polynom der Kategorie *Strictly_Non_Linear* zugeordnet.

Kapitel 3

Katalog dynamischer und regelungstechnischer Modelle

In diesem Kapitel werden die Ergebnisse der Studienarbeit vorgestellt. Angefangen wird mit der Struktur des Kataloges. Danach werden das Klassifikationssystem(s. 3.2), die Modelldokumentation(s. 3.3) und die Modellimplementation(s. 3.4) als wichtige Elemente des Kataloges beschrieben. Eine Auflistung der aktuell vorhandenen Modelle(s. 3.5) und eine Beschreibung des Ablaufes der Katalogerweiterung(s. 3.6) schließen das Kapitel ab.

3.1 Katalogstruktur

Der Katalog besteht aus folgenden Elementen: Dem *Klassifikationssystem*, dem Python Package *GeneralModel* und Modelleinträgen, die aus einer *Metadaten-Datei*, der *Modelldokumentation*, einer *Parameter-Datei* und, optional, aus der implementierten *Modellklasse* bestehen. Für jede Datei der Modelleinträge gibt es eine Vorlage. Die Beziehungen zwischen den Elementen sind in Abbildung 3 dargestellt.

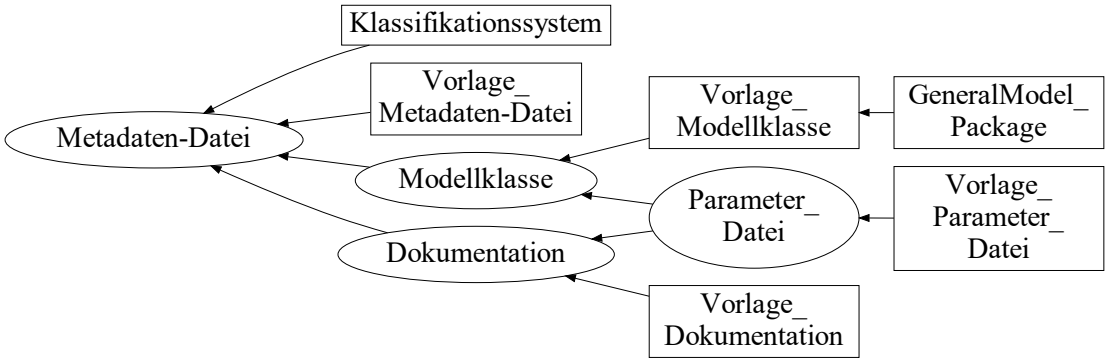

Abbildung 3 – Elemente in Ellipsen existieren individuell für jedes Modell. Elemente in Rechtecken existieren genau ein Mal im Katalog.

Das *Klassifikationssystem* ist eine Übersicht der Modelleigenschaften und deren Beziehungen untereinander. Die Einträge in dem Feld für die Modelleigenschaften in der Metadaten-Datei(*tag_list*) sind nur Namen von Kategorieknoten aus dem KS. Es sorgt für eine einheitliche Namensgebung der Modelleigenschaften(s. Anforderung **A.2**).

Das Python Package *GenericModel* stellt die Python Klasse *GenericModel* zur Verfügung, von der alle Modellklassen der implementierten Modelle erben. Sie stellt ein Variablen- und Methodenset bereit, das alle Modellklassen gemeinsam haben.

Die *Metadaten-Datei* ist eine Datei im YAML-Format, das Informationsfelder für Informationen zu dem zugehörigen Modell enthält. Es gibt Felder für Informationen...

- über das Modell (Modellname, Kurzbeschreibung, Modelleigenschaften, Dateinamen der Modellbilder)
- für eine Umsetzung des Kataloges als Datenbank(Key, Predecessor-Key, Implementation-Key)
- anderer Art (Modellersteller, Erstellungsdatum, Liste der Bearbeiter, Externe Referenzen)

Das Konzept und die Umsetzung der Metadaten-Datei stammt aus dem *ACKRep*[1].

Die *Modelldokumentation* ist die textuelle Notation des Modells. Sie wird im LaTeX-Format geschrieben. Nach jeder Bearbeitung wird die PDF-Datei aus der LaTeX-Datei erzeugt.

Die *Parameter-Datei* ist eine Python-Datei. Diese enthält beispielhafte Parameterwerte für das Modell. Das Ausführen der Datei erzeugt eine Tabelle mit den beispielhaften

[1] ACKRep GitHub Repository: https://github.com/ackrep-org/ackrep_data

Parameterwerten im LaTeX-Format, schreibt diese in eine Datei Namens *parameters.tex* und speichert die Datei im Ordner der Modelldokumentation. Außerdem stellt Sie eine Methode für die implementierte Modellklasse bereit, welche die Parameter des Modells als Rückgabewert hat.

Die implementierte *Modellklasse* ist eine Python-Datei, welche das Modell als Python-Klasse enthält.

Die *Vorlagen* für die Metadaten-Datei, die Parameter-Datei, die Modelldokumentation und -implementation sind Dateien, welche den Arbeitsaufwand für das Anlegen neuer Modelle verringern sollen(siehe Entscheidung **E.7**). Die repetitiven Elemente für die entsprechenden Dateien sind in den Vorlagen schon vorhanden, so dass nur die Modellspezifischen Elemente neu geschrieben werden müssen. Die Verwendung der Vorlagen stellt eine einheitliche Struktur der Dateien sicher.

Die Ordnerstruktur des Kataloges liegt in einem Git-Repositorium[2](s. Entscheidung **E.8**).

3.2 Klassifikationssystem

Das *Klassifikationssystem (KS)* stellt eine Wissensrepräsentation dar. Es lehnt stark an die in [7] eingeführte *OCSE* an, von der es sich insofern unterscheidet, das im KS nur die Teilbereiche des Wissens der Mathematik, Regelungs- und Steuerungstheorie enthalten sind, die sich auf regelungstechnische Systeme und Modelle beziehen. Die im KS verwendeten Bezeichnungen sollen in den Metadaten-Dateien der Modelle bevorzugt verwendet werden.
Es ist anzumerken, dass das KS keine vollständige Wissensrepräsentation darstellt. Das liegt daran, dass das darzustellende Wissen sehr umfangreich ist und eine vollständige Darstellung dessen im Rahmen dieser Studienarbeit nicht schaffbar war. Weiter könnte man auch die Frage stellen, ob überhaupt eine vollständige Darstellung des beabsichtigten Wissensbereiches existiert, da das Wissen dynamisch mit neuen Entdeckungen wächst. Diese Diskussion soll hier aber nicht weitergeführt werden.

Bei einem konkreten Modell im Katalog finden sich die Einträge des KS im Informationsfeld *tag_list* der Metadaten-Datei wieder. In diesem Informationsfeld werden die Modelleigenschaften und dessen Werte notiert, wobei für die Modelleigenschaften die Begriffe aus dem KS bevorzugt verwendet werden.Für die Einträge in der *tag_list* gilt eine Open-World Annahme. Wenn eine Eigenschaft nicht in der *tag_list* enthalten ist, dann enthält der Katalog keine Aussage darüber, ob das Modell die Eigenschaft besitzt oder nicht.

[2]Der Katalog hat kein eigenes Repository, sondern liegt aktuell im Git-Repository dieser Studienarbeit. Deshalb wird an dieser Stelle kein Link zur Verfügung gestellt.

Das KS besteht aktuell aus <Knotenanzahl> Knoten. <Bildreferenz> zeigt ein Ausschnitt des KS.

3.2.1 Aufbau des Klassifikationssystems

Das KS ist ein Semantisches Netz(s. **E.10**), welches durch einen gerichteten, kreisfreien Graphen repräsentiert wird. Es gibt folgende Knotentypen: Kategorie-, Objekt-, Werte, und Wertetypknoten. Die Kanten zeigen Beziehungen zwischen den Knoten des KS auf, die durch die Kantenbeschriftungen spezifiziert werden. Es gibt folgende Kantennamen: „is a“, „Value“, „Type“ und „Object“. Der Kantenname kennzeichnet zudem den Knotentyp des Startknotens der Kante. Jeder Endknoten einer Kante ist ein Kategorieknoten. Die Beziehung zwischen Knotentypen und Kantennamen wird in Tabelle 1 gezeigt.

Startknotentyp	Kantenname	Endknotentyp
Kategorieknoten	is a	Kategorieknoten
Werteknoten	Value	Kategorieknoten
Wertetypknoten	Type	Kategorieknoten
Objektknoten	Object	Kategorieknoten

Tabelle 1 – Beziehung zwischen Kantennamen und Knotentypen

Die Kategorieknoten, außer der Ursprungsknoten und die Knoten der Hauptkategorien, können als Modellattribute interpretiert werden. In der Metadaten-Datei dürfen nur diese Knoten des KS eingetragen werden.
Die Kategorieknoten im KS werden durch ihren eindeutigen Namen identifiziert(s. **E.11**).
Jeder Knoten k, außer der Ursprungsknoten, ist entlang eines gerichteten Pfades $P(k)$ mit dem Ursprungsknoten verbunden. Wird einem Modell ein Knoten k_i des KS als Modellattribut zugewiesen, dann hat das Modell auch alle Knoten, außer Ursprungs -und Hauptknoten, entlang des gerichteten $P(k_i)$ als Modellattribut. Aus diesem Grund wird in der *tag_list* der Metadaten-Datei immer nur die spezifischste Eigenschafte eines Pfades angegeben.

Der Ursprungsknoten *Property_Of_Classification_System* hat die drei Unterkategorien *Property_Of_Mathematical_Representation*, *Model_Behaviour* und *Usage*. Die Unterkategorien des Ursprungsknoten werden im KS als Hauptkategorien bezeichnet, die jeweils verschiedene Teilmengen von Modellattributen enthalten. Die Hauptkategorien werden wie folgt Beschrieben.

Eigenschaften der mathematischen Repräsentation:
Umfasst Eigenschaften der mathematische Repräsentation des Modells.

Modelleigenschaften:

Umfasst Eigenschaften die aus der mathematischen Repräsentation mit Methoden der Regelungstechnik abgeleitet werden.

Verwendung:
Umfasst Aufgabentypen und Anwendungsbereiche in denen die Modelle häufig genutzt werden.

In 2.1 wurde erwähnt, dass ein Modell mehrere mathematische Repräsentationen haben kann. Dieser Aspekt führt auf den Begriff der *Äquivalenz* zweier mathematischer Modelle. Zwei mathematische Modelle heißen *Äquivalent*, wenn beide die gleiche Menge an Trajektorien für ihre externen Variablen zulassen[3]. Oder einfacher, wenn beide Modelle das gleiche dynamische Verhalten aufweisen. Das Modell eines Systems kann durch eine Menge äquivalenter mathematischer Modelle repräsentiert werden. Der Unterschied zwischen den Hauptkategorien *Eigenschaft der mathematischen Repräsentation* und *Modelleigenschaften* liegt darin, dass die Elemente eine Menge äquivalenter mathematischer Modelle die gleichen *Modelleigenschaften* haben, sich aber in ihrer mathematischen Repräsentation unterscheiden.

3.2.2 Technische Umsetzung des Klassifikationssystems

Das KS wird technisch mit vier Dateien im YAML-Format realisiert, welche folgende Namen haben: *KS_Tree_main*, *KS_Tree_Math_Representation*, *KS_Tree_Model_Attributes* und *KS_Tree_Usage*. Die Datei *KS_Tree_main* enthält die *Informationsblöcke(IB)* zu dem Ursprungsknoten und den Knoten der Hauptkategorien. Die restlichen drei Dateien enthalten jeweils die Informationsblöcke zu den Knoten der Unterkategorien der Hauptkategorien.
Ein IB enthält alle Informationen zu genau einem Knoten des KS. Jeder IB in den YAML-Dateien ist ein Mapping. Ein Mapping, gekennzeichnet durch einen Doppelpunkt, setzt einen Schlüssel mit genau einem Wert in Beziehung. Das *dictionary* ist das Python-Äquivalent zu einem Mapping. In jedem IB ist der Name des Knotens der Schlüssel und eine Sequenz ist der Wert. Eine Sequenz ist in YAML eine Folge von Zeilen, die mit einem Strich beginnen. *List* und *tuple* sind die Python-Äquivalente zur Sequenz. Jeder Eintrag der Sequenz steht für ein Attribut des Knotens und ist wiederum ein Mapping, dessen Schlüssel der Attributsname und dessen Wert der Attributswert ist. Die IB‘s sind durch eine Leerzeile voneinander getrennt. Die Abbildung 4 zeigt einige IB‘s aus der Datei *KS_Tree_Math_Representation*.

Eine Kante wird durch die Elemente *Pre_Node* und *Edge_Name* definiert. Diese stehen jeweils als Einträge in der Sequenz des Informationsblocks des Startknotens der Kante.

Zusätzlich gibt es das Python-Skript *KS_Create_Graph*, welches die YAML-Dateien mit

[3] vgl. [17], S. 34

```
General_Function:
        - Type: boolean
        - Pre_Node: Property_Of_Mathematical_Representation
        - Edge_Name: is a

DAE:
        - Type: boolean
        - Pre_Node: General_Function
        - Edge_Name: is a
```

Abbildung 4 – Informationsblöcke des KS

Hilfe des Packages *pyyaml*[4] einliest. Zudem erzeugt es einen gerichteten Graphen des Python Packages *networkx*[5] und zeichnet den Subgraphen, der nur die Kategorieknoten, enthält mit dem Python Package *nxv*[6] in eine Bilddatei.

Ein gerichteter Graph des networkx-Packages besteht aus einer Menge von Knoten und Kanten. Zu jedem Knoten und jeder Kante kann eine beliebig große Menge individuell benannter Attribute hinzugefügt werden. So werden z. B. die Kantennamen der entsprechenden Kante als Attribut hinzugefügt. Die Wertetypknoten sind als Attribut eines Kategorieknotens implementiert. Diese Umsetzung des implementierten Graphen ist aus technischer Sicht sinnvoll, da die Information über die Attribute eines Knotens so intuitiver zu erreichen sind. Es soll hier jedoch erwähnt werden, das diese Implementierung von einer korrekten Darstellung des KS als Semantisches Netz abweicht, da Kategorieknoten nur explizite, als Knoten aufgeführte Attribute besitzen. Für eine formal korrekte Implementierung des KS als Semantisches Netz müssten Knoten welche die Attribute einer Kategorie darstellen auch explizit als Knoten angelegt werden.
Die gewählte Implementierung des KS ist einfach erweiterbar und bearbeitbar. Eine Veränderung im Umfang des KS, also das Hinzufügen neuer Knoten zum KS oder das entferne von Knoten aus dem KS, wird in der Implementierung durch schreiben neuer IB‘s oder entfernen von IB‘s erreicht. Sollen neue Knotentypen zum KS hinzugefügt werden z. B. Referenzknoten, welche eine Referenz zur Definition der Kategorie enthalten würden, so werden diese als neuer Eintrag zu den Sequenzen der IB‘s hinzugefügt.

Die Implementierung des KS ist auch relativ einfach für eine potenzielle Integration einer Suchfunktion zu öffnen, indem zu dem Python-Skript *KS_Create_Graph* eine Methode hinzugefügt wird, welche die networkx-Graphen des KS als Rückgabewert liefert.

[4]Pyyaml Package: https://pyyaml.org/wiki/PyYAMLDocumentation
[5]Networkx Package: https://networkx.org/documentation/stable/index.html
[6]Nxv Package: https://nxv.readthedocs.io/en/latest/index.html

3.3 Modelldokumentation

Die Modelldokumentation enthält die textuelle Notation des Modells in einer Datei im LaTeX-Format. Die Dokumentation besteht aus den Abschnitten: *Nomenklatur*, *Modellgleichungen*, *Herleitung und Erklärungen* und *Referenzen*. Der Abschnitt *Herleitung und Erklärung* ist optional.

Nomenklatur:
Die Nomenklatur enthält alle in der Dokumentation genutzten Variablen und eine Beschreibung, für welche Größen diese stehen. Die Nomenklatur ist aufgeteilt in die Sektionen *Nomenklatur für die Modellgleichungen* und *Nomenklatur für die Herleitung*. Die Nomenklatur für die Modellgleichungen enthält die Variablen, die im gleichnamigen Abschnitt verwendet werden und muss in jeder Dokumentation enthalten sein. Die Nomenklatur für die Herleitung enthält alle Variablen die für den Abschnitt *Herleitung und Erklärung* genutzt werden, bis auf die Variablen die in der Sektion *Nomenklatur für die Modellgleichungen* schon enthalten sind. Die Nomenklatur für die Herleitung ist nur zu schreiben, wenn der Abschnitt *Herleitung und Erklärung* existiert.

Model Documentation of the:

Boost Converter

1 Nomenclature

1.1 Nomenclature for Model Equations

U	Voltage
I	Current
L	Inductivity
C	Capacity
R	Electrical Resistance
U_{DC}	Input DC-Voltage
d	duty ratio

Abbildung 5 – Nomenklatur aus der Dokumentation zum Hochsetzsteller(Boost-Converter)

Modellgleichungen:
Der Abschnitt *Modellgleichungen* besteht aus einer formalisierten Darstellung der Modellgleichungen und weiteren Sektionen, die gleich noch benannt werden. Die formalisierte Darstellung der Modellgleichungen besteht aus drei Teilen.

- einer Zuordnung der Variablen zum generalisierten Zustands-, Eingangs- und (optional) Ausgangsvektor
- einem Gleichungssystem von Differentialgleichungen erster Ordnung(s. **E.3**), das die generalisierten Zustands- und Eingangsvariablen verwendet
- der Benennung, welche Variablen Parameter sind

2 Model Equations

State Vector and Input Vector:

$$\underline{x} = (I\ U)^T$$
$$\underline{u} = d$$

System Equations:

$$\dot{x}_1 = -(1-d)\frac{1}{L}x_2 + \frac{U_{DC}}{L} \tag{1a}$$
$$\dot{x}_2 = (1-d)\frac{1}{C}x_1 - \frac{1}{RC}x_2 \tag{1b}$$

Inputs: d
Parameters:$R,\ C,\ L,\ U_{DC}$
Outputs: U

Abbildung 6 – Gleichungen aus der Dokumentation zum Hochsetzsteller(Boost-Converter)[7]

Die weiteren Sektionen sind recht frei gestaltbar und sollen weitere nützliche Informationen zu dem Modell liefern. Aktuell sind in der Vorlage nur die Sektionen *Annahmen* und *Beispielhafte Parameterwerte* explizit formuliert. Weitere Sektionen können vom Ersteller des Modells nach eigener Einschätzung der Sinnhaftigkeit hinzugefügt werden. Denkbar wäre, angenommen es handelt sich um ein flaches Modell, z. B. ein Abschnitt der flache Ausgänge benennt. Die Sektion *Beispielhafte Parameterwerte* ist als einzige unter den weiteren Sektionen zwingend zu füllen. Dies geschieht mehr oder weniger automatisch, da die Datei *paramters.tex*, die durch die *Parameter-Datei* erstellt wird, dafür eingebunden wird.

Herleitung und Erklärungen:
Dieser Abschnitt ist für die Dokumentation optional. Er soll die Möglichkeit bieten die Herleitung des Modells zu beschreiben oder weitere aus Sicht des Modellerstellers sinnvolle Erklärungen zu liefern.

[7] Modell stammt aus [15] Abschnitt 1.3

Referenzen:
Enthält die Referenzen, die für die Erstellung der Modelldokumentation verwendet wurden.

Aus der LaTeX-Datei der Modelldokumentation wird eine für Menschen gut lesbare PDF-Datei erzeugt.

3.4 Modellimplementation

Modelle des Kataloges werden in Form einer Python-Klasse implementiert. Bei Initialisierung eines Objektes der Modellklasse können dem Konstruktor die Zustandsdimension, eine Eingangsfunktion und ein Parametervektor übergeben werden. Alle dieser Variablen haben Standardwerte. Jede Modellklasse bezieht ein Standardset von Parameterwerten aus der *Parameter-Datei* (die Standardparameter sind also identisch mit den beispielhaften Parameterwerten aus der Dokumentation) und hat eine Standardfunktion für die Modelleingänge integriert.
Ein Modell kann also einfach über ein eigens erstelltes Python-Skript simuliert werden(vgl. **A.5**), indem eine Objekt der Modellklasse erzeugt wird, ein Startvektor für den Modellzustand definiert wird und anschließend das Modell mit einer geeigneten Methode(z. B. *solve_ivp* des Packages *scipy*) simuliert wird. Die Methode *get_rhs_func* der Modellklasse stellt die dafür benötigte Python-Funktion zur Verfügung.

Die Parameter und die Eingangsfunktion eines Modellobjektes können nach Initialisierung mit den Methoden *set_parameters* und *set_input_func* geändert werden. Die Zustandsdimension ist nur bei Initialisierung einstellbar und das auch nur bei Modellen, deren Zustandsdimension nicht festgelegt ist(z. B. beim N-Fach-Integrator).
Die Methode *get_rhs_symbolic* liefert die symbolischen Modellgleichungen als Rückgabewert. Zur Implementierung der symbolischen Gleichungen wird das Python-Package *sympy* verwendet.

Jede Modellklasse erbt von der Klasse *GenericModel*, welche eine Menge von Methoden und Variablen definiert, die jede Modellklasse haben muss. Die Methoden, die nicht für jede individuelle Modellklasse angepasst werden müssen sind in der Klasse *GenericModel* implementiert.
Innerhalb einer Modellklasse sind nur die symbolischen Gleichungen als Code geschrieben. Die Umwandlung in die numerischen Form erfolgt in der Methode *get_rhs_func* mit der Methode *lambdify* des Sympy-Packages.

Mittels der aktuellen Vorlagen können nur Modelle, deren Verhalten durch gewöhnliche Gleichungssysteme erster Ordnung beschrieben wird, implementiert werden.

3.5 Vorhandene Modelle

3.6 Katalogerweiterung

Die letzte Anforderung an den Katalog war, das dieser erweiterbar sein soll (Anforderung **A.6**). In diesem Abschnitt wird beschrieben, welche Schritte für die Erweiterung des Kataloges nötig sind.

Der Katalog wird über das Hinzufügen neuer Modelle erweitert. Wir treffen Annahmen:

1. Die Recherche für das hinzuzufügende Modell ist abgeschlossen. Die nötigen Informationen zu dem Modell wie Modellgleichungen, beispielhafte Parameterwerte, Beschreibung der Variablen und die dazugehörigen Referenzen sind also schon bekannt.

2. Das Modell wird ausschließlich durch gewöhnliche Differentialgleichungen beschrieben.

3. Der Modellersteller hat sich mit der Ordnerstruktur des Kataloges vertraut gemacht.

Für das Anlegen eines neuen Modells im Katalog sind folgende Schritte notwendig:

1. Lokale Installation von Git und Herunterladen des Git-Repositoriums(siehe **E.8**) um eine lokale Kopie des Kataloges und der Vorlagen(siehe Abschnitt 3.1 Katalogstruktur) zu erhalten.

2. Anlegen und Benennung eines neuen Modell- und Implementationsordners.

3. Erstellen der *Metadaten-Datei* im Modellordner mithilfe der Vorlage und Umbenennung dieser in *metadata.yml*.

4. Erstellen der *Modelldokumentation* im Modellordner mithilfe der Vorlage.

5. Erstellen der *Parameter-Datei* im Implementationsordner mithilfe der Vorlage und ausführen[8] dieser.

6. PDF-Datei der Modelldokumentation erzeugen.

7. (optional, siehe **E.5**) Erstellen der *Modellklasse* im Implementationsordner mithilfe der Vorlage.

8. Pull-Request ausführen.

[8] Zur Erinnerung: Die Parameter-Datei ist ein Python Skript.

Kapitel 4

Zusammenfassung und Ausblick

4.1 Übersicht zu aktuellem Funktions- und Modellumfang

Inhalt:
Aktueller Stand bzgl. Modellumfang und möglicher Modellkomplexität –> Problem mit komplexen Größen
Aktuelle Vorlagen und Limitierungen, Aufwandseinschätzung für das Hinzufügen neuer Modelle

4.2 Ausblick

Inhalt:
Was noch denkbar/wünschenswert/möglich wäre bzgl. Funktionsumfang
Versionsverwaltung, Ideen/Konzept zur Öffnung für breite(re) Nutzerschaft(Mögliche Vorlage: ACKRep, aber evtl. Orientierung auch an anderen Nutzer basierten Datenbanken (Wiki-like))
Abfragbare KS Implementierung via networkx-Package die weitere Informationen zu Einträgen des KS enthält (kurze Beschreibung, Referenz bzgl. Bedeutung –> Umsetzung mit .bib Datei und Verweis auf Kürzel + ggf. Seitenangabe etc.)
Suchfunktion
Automatische Erstellung einer Übersicht aller der enthaltenen Modelle (ist Verknüpft mit Suchfunktion?)

Literatur

[1] Alexander Benevolenskiy. *Ontology-based modeling and configuration of construction processes using process patterns = Ontologie-basierte Modellierung und Konfiguration der Bauprozesse mit Hilfe von Prozessvorlagen*. Dresden: Institut für Bauinformatik, Fakultät Bauingenieurwesen, TU Dresden, 2016. ISBN: 9783867804776.

[2] Eugene I. Butikov. „Kapitza's Pendulum: A Physically TransparentSimple Treatment“. Web Link: http://butikov.faculty.ifmo.ru/InvPendulumCNS.pdf. 2021.

[3] Reinhard Diestel. *Graph theory*. New York: Springer, 2000. ISBN: 0387989765.

[4] DIN IEC 60050-351:2014-09, Deutsches Institut für Normung e. V. *Internationales Elektrotechnisches Wörterbuch –Teil 351: Leittechnik*. 2014.

[5] Hendrik Fehr und Albrecht Gensior. „Improved Energy Balancing of Grid-Side Modular Multilevel Converters by Optimized Feedforward Circulating Currents and Common-Mode Voltage“. In: *IEEE Transactions on Power Electronics* 33.12 (Dez. 2018), S. 10903–10913. DOI: 10.1109/tpel.2018.2805103.

[6] Gilbert Greefrath und Katrin Vorhölter. *Teaching and Learning Mathematical Modelling*. Springer International Publishing, 2016. DOI: 10.1007/978-3-319-45004-9.

[7] Carsten Knoll; Robert Heedt. „Tool-based Support for the FAIR Principles for Control Theoretic Results: "The Automatic Control Knowledge Repository"“. [Soll im: System Theory, Control and Computing Journal veröffentlicht werden.] 2020.

[8] Carsten Knoll. „Regelungstheoretische Analyse- und Entwurfsansätze für unteraktuierte mechanische Systeme“. Diss. Technische Universität Dresden, Juni 2016.

[9] Carsten Knoll und Robert Heedt. „“Automatic Control Knowledge Repository” – A Computational Approach for Simpler and More Robust Reproducibility of Results in Control Theory“. In: *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE, Okt. 2020. DOI: 10.1109/icstcc50638.2020.9259657.

[10] Yang Liu und Hongnian Yu. „A survey of underactuated mechanical systems“. In: *IET Control Theory & Applications* 7.7 (Mai 2013), S. 921–935. DOI: 10.1049/iet-cta.2012.0505.

[11] Edward N. Lorenz. „Deterministic Nonperiodic Flow“. In: *Journal of Atmospheric Sciences* 20 (März 1963), S. 130–141.

[12] Günter Ludyk. *Theoretische Regelungstechnik 1*. Springer Berlin Heidelberg, 1995. ISBN: 978-3-540-55041-9. DOI: 10.1007/978-3-642-77221-4.

[13] Jan Lunze. *Regelungstechnik 1*. Springer Berlin Heidelberg, 2010. ISBN: 978-3-642-13807-2. DOI: 10.1007/978-3-642-13808-9.

[14] Sharmila J. Petkar u. a. „Robust Model Predictive Control of PVTOL Aircraft“. In: *IFAC-PapersOnLine* 49.1 (2016), S. 760–765. DOI: 10.1016/j.ifacol.2016.03.148.

[15] Klaus Röbenack. *Nichtlineare Regelungssysteme*. Springer Berlin Heidelberg, 2017. DOI: 10.1007/978-3-662-44091-9.

[16] Otto E. Rössler. „Continuous Chaos - Four Prototype Equations“. In: *New York Acadamy of Sciences* (1979), S. 376–392.

[17] A. J. van der Schaft. „Transformations of nonlinear systems under external equivalence“. In: *New Trends in Nonlinear Control Theory*. Springer Berlin Heidelberg, 1989, S. 33–43. DOI: 10.1007/bfb0043015.

[18] J. Sebestyénová. „Usage of Domain Ontology in e-Learning“. In: (2004).

[19] Alecsandru Simion, Leonard Livadaru und Adrian Munteanu. „Mathematical Model of the Three-Phase Induction Machine for the Study of Steady-State and Transient Duty Under Balanced and Unbalanced States“. In: *Induction Motors - Modelling and Control*. InTech, Nov. 2012. DOI: 10.5772/49983.

[20] Herbert Stachowiak. *Allgemeine Modelltheorie*. Wien, New York: Springer-Verlag, 1973. ISBN: 3211811060.

[21] Heiner Stuckenschmidt. *Ontologien*. Springer Berlin Heidelberg, 2009. DOI: 10.1007/978-3-540-79333-5.

[22] Rudi Studer, V.Richard Benjamins und Dieter Fensel. „Knowledge engineering: Principles and methods“. In: *Data & Knowledge Engineering* 25.1-2 (März 1998), S. 161–197. DOI: 10.1016/s0169-023x(97)00056-6.

[23] Sansal K. Yildiz u. a. „Dynamic modelling and simulation of a hot strip finishing mill“. In: *Applied Mathematical Modelling* 33.7 (Juli 2009), S. 3208–3225. DOI: 10.1016/j.apm.2008.10.035.