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:	20. 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

— Noch zu verfassen. —

1.2 Präzisierung der Aufgabenstellung

— Noch zu verfassen. —

Kapitel 2

Vorbetrachtung

Der in Kapitel 3 vorgestellte Katalog und dem dazugehörigen Klassifikationssystem ist das Resultat vieler Entscheidungen. Dem ist eine Recherche von Literatur mit regelungstechnischen Modellen vorausgegangen. Das Kapitel beginnt mit einer Erklärung zu den Begriffen *System* und *Modell*. Anschließend wird in Abschnitt 2.2 der aktuelle Stand der Modellrecherche und vorhandener Modellsammlungen vorgestellt. Danach werden in Abschnitt 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 Abschnitt 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 *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 ([4, S. 21]) ausgeführt, das die Elemente eines Systems „natürliche oder künstliche Gegenstände sowie Arten von Denkvorgängen und deren Ergebnisse“ sein können und dass das System „als von der Umgebung und von den anderen äußeren Systemen durch eine gedachte Hüllfläche

[1] siehe [4, S. 21]

abgegrenzt betrachtet“ wird.

Ein System wird in dieser Arbeit als *reales System* bezeichnet, wenn dessen Elemente nachweisbare Bestandteile der Realität sind.

Modelle sind (nach Stachowiak[19, S. 13ff]) 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*).

Modelle von Systemen(*Systemmodelle*) bestehen in der Regelungstechnik zum einen aus einer grafischen Abbildung des Systems, die wiederum aus definierten Formen besteht, welche die Elemente des Systems repräsentieren und zum anderen aus einem mathematischen Modell, welches das Verhalten des Systemmodells beschreibt. Ein System kann mehrere Modelle in Form einer graphischen Repräsentation haben, wobei es zu jedem Modell wiederum verschiedene mathematische Modelle geben kann(vgl. [11, Abschnitt 2.1]).
Ein *mathematisches Modell* ist ein Modell, das die Anwendung mathematischer Methoden erlaubt([5], S.9). Das mathematische Modell ist in der Regelungstechnik üblicherweise ein System von Gleichungen.

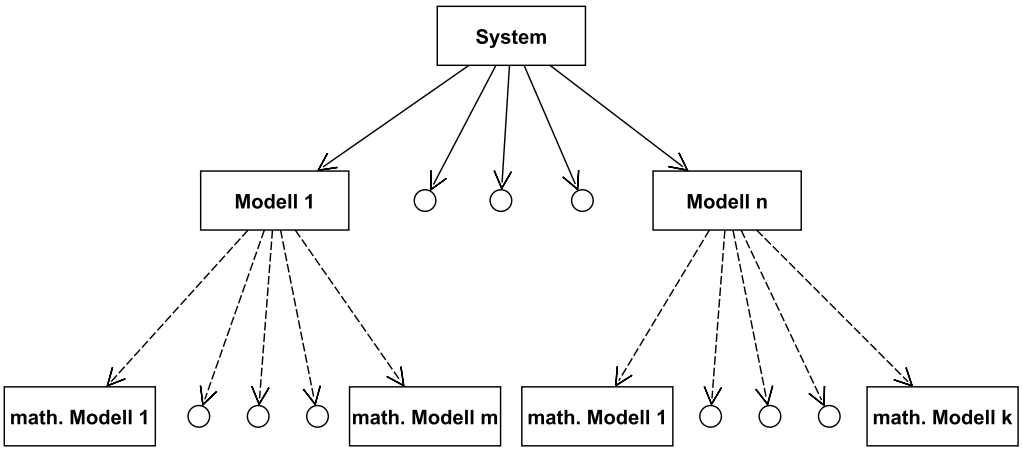


Abbildung 1 – Zusammenhang Systeme, Modelle und mathematische Modelle

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.

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.

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 der Modellerschließung

Dieser Abschnitt beschreibt die Erkenntnisse aus einer umfangreichen Literaturrecherche bezüglich Publikationen, die regelungstechnische Modelle enthalten. Folgende Erkenntnisse lassen sich festhalten:

Aktuelle Situation der Modellrecherche:

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. Dass die Modellgleichungen eindeutig gekennzeichnet und gemeinsam notiert, sowie die eingeführten Variablen gut beschrieben und klar definierten Typs (Parameter, Eingangs-, Zustandsvariable) sind, ist nicht immer gegeben.

    - Beispiel 1: In [10] wird auf Seite 135 das Modell eines dynamischen Systems (bekannt als sog. „Lorenz-System“) 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 [22] wird eine Walzstrecke für Metalle modelliert. Die Variablen werden am Anfang alle eingeführt und danach wird das Modell 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 [18] Seite 14 werden die Gleichungen des Modells als Gleichungs-

system von gewöhnlichen Differentialgleichungen(DGLn) erster Ordnung dargestellt. Die Gleichungen sind jedoch nicht wie üblich umgestellt[2], denn auf der linken Seite der Gleichungen befinden sich Summen aus Zeitableitungen und nicht-abgeleiteten Zustandsgrößen.

- Beispiel 4: In [2] Seite 3 besteht das mathematische Modell des Systems aus einer gewöhnlichen DGL zweiter Ordnung
- Beispiel 5: In [7] Seite 168f, Beispiel B.3 besteht das mathematische Modell des betrachteten Systems aus einem gewöhnlichen Differentialgleichungssystem zweiter Ordnung, wobei wie in Bsp. 3 diese nicht wie üblich umgestellt worden sind.

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 in der Publikation untersuchte Fragestellung von Relevanz sind.
   - Beispiel 6: Im Artikel [13] 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] wird der Verlauf der Eingangswerte, welcher für die im Dokument gezeigten Simulationsergebnisse verwendet wurde, nur als grauer Graph gezeigt. Eine Angabe der Gleichung für die Eingangswerte fehlt. Ebenso wird z. B. der verwendete Parameterwert für die Gleichspannung $v_D C$ nicht gegeben.
6. Die genutzte Implementation wird nicht veröffentlicht.

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 eines symbolischen Analysemodells und ausführbaren Simulationsmodells muss aktuell fast immer von eigener Hand erfolgen. Für die Validierung des eigenen Codes und die Reproduktion der Resultate einer Publikation ist oft eine softwaretechnische Implementation des Modells sowie der daran angehängten Umgebung (Steuerung, Regelung, Beobachter etc.) notwendig (vgl. [8], Seite 1). Durch die oben beschriebenen Erkenntnisse zur Modellrecherche ist das meist aufwendig oder nicht möglich.

[2] Üblicherweise werden die Modellgleichungen so umgestellt, das sich die Zeitableitungen der Zustände möglichst allein auf der linken Seite und das sich die restlichen Variablen und nicht-abgeleiteten Zustände auf der rechten Seite der Gleichung befinden.

Aktuelle Situation von Modellsammlungen und -katalogen:
Bestehende Zusammenstellungen von regelungstechnischen Modellen sind schwer zu finden. Ein Grund dafür könnten sein, das wenige existieren. Es könnte aber auch daran liegen, das diese einfach nur schlecht zu auffindbar sind. Zum Beispiel, weil diese von der Suchmaschine als nicht relevant eingestuft werden und dementsprechend auf den hinteren Seiten der Suchergebnisse landen. Zudem basieren die Suchergebnisse nur auf den eingegebenen Wörtern. Ein Verständnis für den Kontext fehlt, was zu einer großen Anzahl an Suchergebnissen führt. Es ist also durchaus plausibel, das mehr als die hier aufgeführten Zusammenstellungen existieren.

The Automatic Control Knowledge Repository (ACKRep):
Das in [6] vorgestellte ACKRep[3] ist ein Tool, mit dem Ziel die implementierten Ergebnisse von Publikationen reproduzierbarer zu machen. Es besteht aus einer Datenbank von implementierten Problemspezifikationen, Lösungsmethoden und Softwareumgebungen, die in ihrem Zusammenwirken die Ergebnisse reproduzierbar machen sollen. Die Einträge vom Typ *ProblemSpecification* enthalten dabei Modelle, die als Python-Quellcode repräsentiert werden bzw. implementiert sind. Weitere Informationen zu den Modellen wie der Identifikationsschlüssel, externe Referenzen und Modelleigenschaften sind in einer Metadaten-Datei im YAML-Format hinterlegt. Für die einheitliche Benennung der Modelleigenschaften wird auf die in [8] eingeführte *Ontology of Control Systems Engineering (OCSE)*[4] zurück gegriffen.

Beispiele unteraktuierter mechanischer Systeme in [9]:
Im Zuge der Betrachtung unteraktuierter mechanischer Systeme werden in dem Artikel die Modellgleichungen von 11 Beispielen tabellarisch in einer vereinheitlichten Form aufgeführt, deren mathematisches Modell aus gewöhnlichen DGLn zweiter Ordnung besteht. Zudem werden die Systeme bezüglich ihrer Beschränkungen, ihrer Konfigurationscharakteristik und ihrer regelungstechnischen Problemstellung klassifiziert.

Die beiden Zusammenstellungen haben einen unterschiedlichen Fokus.
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 eine gefilterte Ausgabe der Elemente ermöglichen soll.
In [9] liegt der Fokus auf der menschlichen Lesbarkeit. Die Modellgleichungen sind als Text bzw. Formeln repräsentiert und ermöglichen die Anwendung von analytischen Methoden.

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

[3] ACKRep Testinstanz. URL: https://testing.ackrep.org/
[4] Die *OCSE* ist eine Wissensrepräsentation(siehe Abschnitt 2.4) zur Regelungs- und Steuerungstheorie.

2.3 Anforderungen an den Katalog

Der Katalog soll den Prozess der Modellrecherche und Nutzung vereinfachen, sodass die in Abschnitt 2.2 beschriebenen Schwierigkeiten vermieden werden. Daher werden für den Katalog folgende Anforderungen formuliert.

Anforderung A.1: Die Modelle des Kataloges 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 eine übersichtliche Darstellungsform haben.
- Sie sollen einer einheitlichen Namensgebung folgen.
- Sie sollen eine 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 sowohl für die symbolische Analyse als auch für Simulationen einfach verwendbar sein.

Anforderung A.6: Der Katalog soll erweiterbar sein.

Die Anforderung **A.6** macht es notwendig den Fall einer zukünftig großen Anzahl im Katalog existierender Modelle mitzudenken. Mit zunehmender Modellanzahl wird es aufgrund abnehmender Übersichtlichkeit schwieriger, bestimmte Modelle im Katalog zu finden. Dazu kommt, das es auch wünschenswert ist, nach Modellen suchen zu können, die bestimmte Kriterien erfüllen. Das macht eine Suchfunktion erstrebenswert um die Anforderung **A.1** zu erfüllen. Um eine Suchfunktion zu realisieren wird eine Datenbank benötigt, die durchsucht werden kann. Die Erstellung einer Suchfunktion und der dafür benötigten Datenbank soll durch folgende Entscheidung erleichtert werden:

Entscheidung E.1: Zu jedem Modell soll es 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** orientiert sich an dem Vorgehen in [6].

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)* (siehe Abschnitt 2.5 und Abschnitt 3.2) und die Anwendung dessen auf die Modelle erfüllt werden. Die Auflistung der Modelleigenschaften eines Modells erfolgt in der Metadaten-Datei. Die Verwendung der im KS verwendeten Namen für die Modelleigenschaften sorgt für eine einheitliche Namensgebung. Die Definition der Modelleigenschaften und der Beziehungen zwischen diesen basieren auf im KS enthaltenen Referenzen. Die folgende Entscheidung befasst sich mit der technischen Umsetzung des KS.

Entscheidung E.2: Die Einträge des KS werden im YAML Format gespeichert. Um eine grafische Darstellung des KS zu erhalten soll ein Python-Skript erstellt werden, welches diese als Bilddatei des KS ausgibt.

Wie vollständig die in der Metadaten-Datei enthaltenen Modelleigenschaften sind, hängt davon ab, wie gut erforscht das Modell ist[5] und wie viel Zeit und Sorgfalt in das Anlegen des Modells investiert wird.

In Textform, also als formatierter Text, dargestellte Modelle haben den Vorteil, dass in Formeln verwendete Elemente wie Tiefsetzungen und Brüche grafisch dargestellt werden können. Im unformatierten Text hingegen findet keine grafische Darstellung von Textelementen statt und die Umsetzung der Elemente in Formeln muss über ein Notationssytem erreicht werden, welches die Lesbarkeit verringert (1/3 statt $\frac{1}{3}$ oder F_1 statt F_1). Eine für Menschen gut lesbare Notation der Modellgleichungen macht die Arbeit mit diesen einfacher und ist generell erstrebenswert, weshalb folgende Entscheidung getroffen wurde.

Entscheidung E.3: Die Modellgleichungen und die zugehörigen Variablen werden als formatierter Text notiert. Grundlage ist ein LaTeX-Dokument, das in eine PDF-Datei umgewandelt wird. Die Modellgleichungen werden als System von gewöhnlichen Differentialgleichungen erster Ordnung dargestellt, insofern dies ohne Informationsverlust möglich ist[6]. Die Auflistung der Variablen und ihrer Bedeutung erfolgt gebündelt.

Die Bündelung der Variablendefinitionen innerhalb der Textrepräsentation macht diese gut im Dokument auffindbar und erfüllt Anforderung **A.4**. Die Festlegung auf die Verwendung von DGLn erster Ordnung ist zum einen recht allgemeingültig, da DGLn höherer Ordnung durch Einführung definitorischer Gleichungen in ein System von DGLn

[5] Damit ist gemeint, das es einfach noch nicht bekannt ist, ob ein Modell bestimmte Eigenschaften hat oder nicht.

[6] Das ist z. B. bei Differenzialalgebraischen Gleichungen oder partiellen Differentialgleichungen nicht möglich. Darauf wird nochmal in Abschnitt 4.2 eingegangen.

erster Ordnung überführt werden können, und zum anderen die in Publikationen am häufigsten verwendete Form der Darstellung der Modellgleichungen. Zudem wird durch diese Festlegung die Anforderung **A.3** erfüllt.

Anforderung **A.5** fordert die Implementierung der Modelle. Darauf bezieht sich die nächste Entscheidung.

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 der Modellklasse instanziiert, was für das Verständnis eine einfache und intuitive Analogie zur realen Welt ist in der reale Systeme als physische Objekte existieren. Die Modellklasse ist in dieser Analogie der Bauplan und das instanziierte Objekt ein spezifischer Aufbau des Systems. Zudem können innerhalb des instanziierten Objektes eine Reihe von Informationen zu dem spezifisch implementierten Modell gespeichert werden. Die Verwendung für eigene Simulationen ist damit sehr unkompliziert, da nur noch der Code für die Simulationsparameter sowie die Aufzeichnung und grafische Darstellung der Simulationsergebnisse selbst geschrieben werden muss. Die für die Simulation benötigten Modellgleichungen können über die Methoden der Klasse direkt verwendet werden.

Es stellte sich noch die Frage, ob alle Modelle des Kataloges implementiert vorliegen müssen. Denn zum Einen ist die in Anforderung **A.3** geforderte einheitliche Darstellung der Modellgleichungen an sich schon durch die in Entscheidung **E.3** beschriebene Textform gegeben. Zum Anderen sind einige Modelle recht schwer zu implementieren, wie z. B. Modelle deren Verhalten durch partielle Differentialgleichungen beschrieben werden. Um eine hohe Modellvielfalt für den Katalog zu ermöglichen, wurde die folgende Entscheidung getroffen.

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

Um die Verwendbarkeit der vorliegenden Modelle weiter zu vereinfachen erscheint es sinnvoll, ein Beispielsatz der Parameterwerte zur Verfügung zu stellen. Die Beispielwerte sollen in der Textform und in der Implementation vorliegen.

Entscheidung E.6: Zu jedem Modell soll es ein Beispielsatz an Parameterwerten geben. Diese sollen sowohl in der Textform als auch in der implementierten Form stets identisch sein. Dafür sollen die Beispielwerte an einer zentralen Stelle notiert sein, von der Textform und Implementation die Werte beziehen.

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 müssen die Vorlagen lokal vorliegen, um neue Modelle anlegen zu können.

Entscheidung E.8: Der Katalog soll 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 ist üblicherweise ein Dateisystem bestehend aus Ordnern und Dateien. Ein Git-Repo kann als lokale Kopie heruntergeladen und lokal editiert werden. Die lokal vorgenommenen Änderungen eines Git-Repos können auf das originale oder offizielle Git-Repo übertragen werden. Das geschieht mittels einer Anfrage zur Übertragung der Änderungen (*Pull-Request* oder *Merge-Request*), welche von den Inhabern des offiziellen Repos geprüft wird. Wird ein Pull-Request angenommen, so werden die im lokalen Repo vorgenommenen Änderungen auf das offizielle Repo übertragen (vgl. [6, 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 den Bestandteilen werden definiert (vgl. [1] S.60, [17] 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*[7]
Ein *Graph* besteht aus einer Menge von *Knoten* N und *Kanten* E. Die Kanten verbinden die Knoten. Die Menge E besteht also aus 2-elementigen Untermengen von N. Ein Element aus E enthält also zwei Knoten, die durch eine Kante verbunden werden. Ist

[7] siehe [20] S.29 und [3] Abschnitte 1.1, 1.3 und 1.10

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 n eines Graphen G über eine Sequenz von Kanten $(e_1, \ldots, e_i)$ verbindet. Ein Knoten kann maximal ein Mal entlang eines Pfades vorkommen. Ein erwähnenswerter Spezialfall ist der kreisfreie Graph. „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“[8].

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“[9]. Im Unterschied zur Taxonomie kann es für die Beziehung zwischen zwei Knoten verschiedene Beziehungstypen geben, welche durch verschiedene Kantentypen beschrieben werden. 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 sich wie in der Taxonomie in Ober- und Unterkategorien aufteilen. Die Kanten zwischen Knoten, die Kategorien darstellen, sind wie bei der Taxonomie mit „ist ein“ bezeichnet. Knoten in Semantischen Netzen können zudem Werte und konkrete Objekte bestimmter Kategorien repräsentieren. Semantische Netze haben häufig eine *Open-World* Annahme. Das bedeutet, 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 ist aber nicht einheitlich definiert. Nach Studer [21] 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.

[8] 20.
[9] [20] S.28

Eine Diskussion zu den Schwächen dieser Definition führt Stuckenschmidt in [20] 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.

In dieser Arbeit wird häufig davon gesprochen, dass das KS auf ein Modell angewendet wird. Damit ist gemeint, dass geprüft wird welche Eigenschaften aus dem KS das Modell besitzt.

Die Frage: „Welchen Umfang soll das KS haben?“ ist dabei eine zentrale Frage, 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** ist: Sollen selten verwendete und eher unbekannte Eigenschaften mit in das KS aufgenommen werden? Dafür spricht, 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, für die Notation der Modelleigenschaften aktiv verwendet zu werden. Die Komplexität des KS niedrig zu halten erscheint dafür ein sinnvoller Gedanke zu sein. Eine Entscheidung zu dieser Frage wurde 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 des KS gegeben. Die folgende Entscheidung fasst diese zusammen.

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

Die Namensgebung ist ein zentrales Element im KS, da diese in den Metdaten-Dateien der Modelle als einheitliche Bezeichnung für die Modelleigenschaften dienen sollen. Dazu wurde die folgende Entscheidung getroffen.

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.

In seltenen Fällen kann es für dieselbe Eigenschaft verschiedene Begriffe in der Literatur geben. In einem solchen Fall soll so entschieden werden, dass der in der Praxis geläufigere Begriff verwendet wird. Ein Beispiel für einen solchen Fall sind die Begriffe *Zeitvarianz* und *Zeitvariabilität*(s. [12], S. 114[10]). Beide bezeichnen die Eigenschaft eines Modells, dessen Modellgleichungen explizit von der Zeit abhängig sind. Perspektivisch kann für solche Fälle, das Element *synonym* in die Menge der Kantennamen aufgenommen werden.
Im Folgenden 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 auch Kategorien sein.

Dass 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.

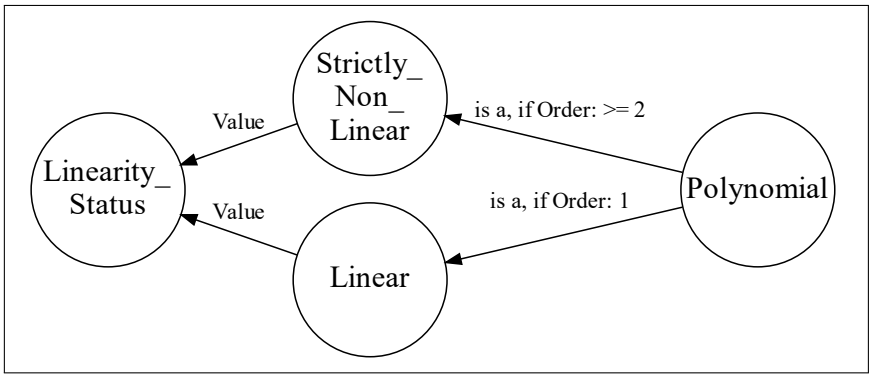
Problemstellung: **Polynom**
Ein Polynom kann abhängig von seiner Ordnung linear oder nichtlinear sein. Um diesen Sachverhalt vollständig im KS abzubilden müsste das Polynom im KS ein Objekt der Kategorien *Linear* und *Strictly_Non_Linear* sein (s. Abbildung 2a), welche zwei Werte der Kategorie *Linearity* sind, die sich gegenseitig ausschließen. In der Anwendung auf

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

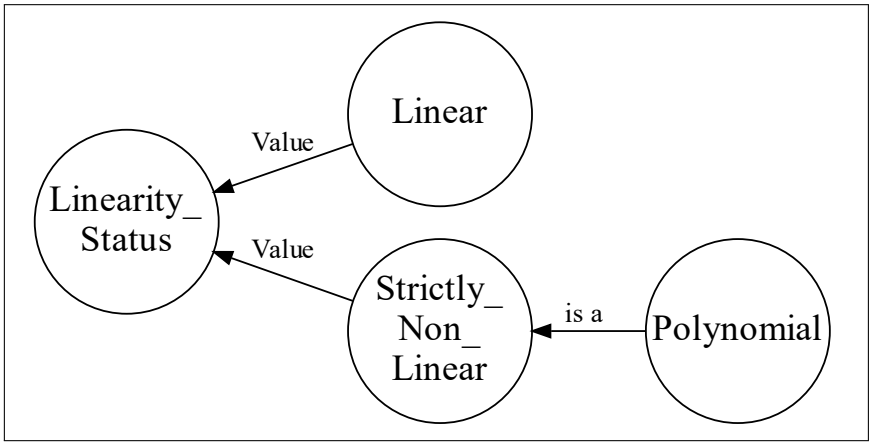
ein Modell lässt sich dieser Widerspruch im Allgemeinen problemlos aufheben, da bei einem spezifischen Modell, dessen Modellgleichungen eine polynomiale Form haben, die Ordnung der Polynome festgelegt ist. Die durch diese Problemstellung aufgetretene Frage ist also: Sollen Widersprüche im KS auftreten dürfen, wenn diese sich bei Anwendung auf ein konkretes Modell 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, um das KS intuitiv verständlich zu halten. In Folge dieser Entscheidung wurde das Polynom der Kategorie *Strictly_Non_Linear* zugeordnet (s. Abbildung 2b).



a) Verworfene Integrationsart für die Kategorie *Polynom* im KS



b) Genutzte Integration der Kategorie *Polynom* im KS

Abbildung 2 – Integrationsmöglichkeiten der Kategorie *Polynom*

Kapitel 3

Katalog dynamischer und regelungstechnischer Modelle

In diesem Kapitel wird der erstellte *Katalog dynamischer und regelungstechnischer Modelle* vorgestellt. Zuerst wird die Struktur des Kataloges gezeigt. Danach werden das Klassifikationssystem (s. Abschnitt 3.2), die Modelldokumentation (s. Abschnitt 3.3) und die Modellimplementation (s. Abschnitt 3.4) als wichtige Elemente des Kataloges beschrieben. Eine Auflistung der aktuell vorhandenen Modelle (s. Abschnitt 3.5) und eine Beschreibung des Ablaufes der Katalogerweiterung (s. Abschnitt 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*. Für jede Datei eines Modells gibt es eine Vorlage. Die Beziehungen zwischen den Elementen sind in Abbildung 3 dargestellt.

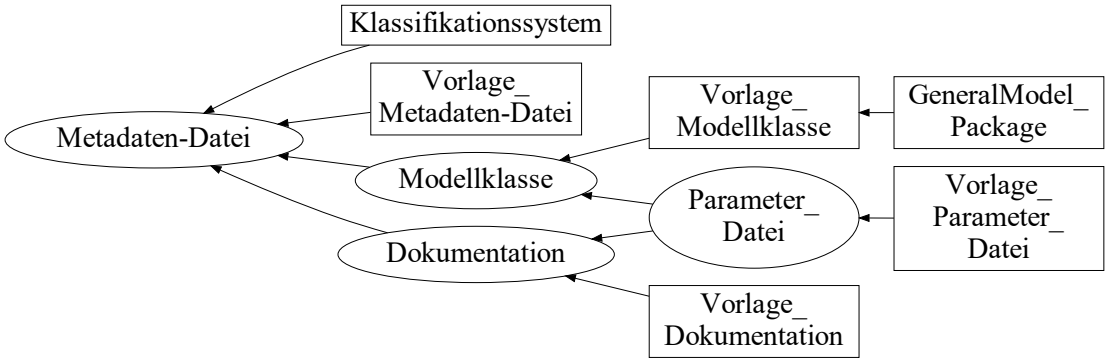


Abbildung 3 – Katalogstruktur. Elemente in Ellipsen existieren individuell für jedes Modell. Elemente in Rechtecken existieren genau ein einziges Mal im Katalog.

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 bevorzugt 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 Variablen und Methoden bereit, die alle Modellklassen gemeinsam haben.

Die *Metadaten-Datei* ist eine Datei im YAML-Format, die Felder für Informationen enthält, in die Informationen des zugehörigen Modells eingetragen werden können. Es gibt Felder für Informationen ...

- über das Modell (Modellname, Kurzbeschreibung, Modelleigenschaften, Dateinamen eventueller 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 basieren auf [8].

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 Parameterwerten im LaTeX-Format und speichert diese in einer Datei Namens *parameters.tex* im Ordner der Modelldokumentation ab. 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, sodass nur die Modell-spezifischen Elemente neu geschrieben werden müssen. Die Verwendung der Vorlagen stellt außerdem eine einheitliche Struktur der Dateien sicher.

Die Ordnerstruktur des Kataloges liegt in einem Git-Repositorium[1](s. Entscheidung **E.8**).

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

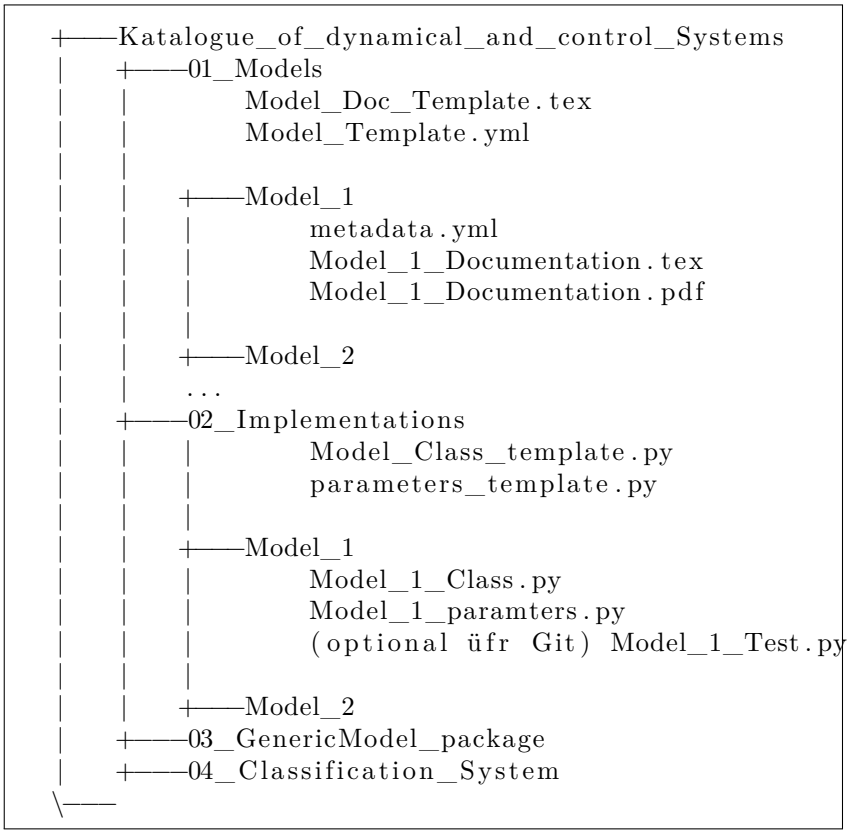


Abbildung 4 – Ordnerstruktur des Git Repositoriums des Kataloges mit schematischer Inhaltsdarstellung der Unterordner für die Modelle und die Implementationen.

3.2 Klassifikationssystem

Das *Klassifikationssystem (KS)* stellt eine Wissensrepräsentation dar. Es lehnt stark an die in [6] 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, da diese nur mit sehr hohem Aufwand erstellbar ist. Zum Einen, weil das potenziell darzustellende Wissen sehr umfangreich ist und zum Anderen, weil aus Sicht des Autors eine vollständige Darstellung des beabsichtigten Wissens, aufgrund der dynamischen Erweiterung des Wissens durch neue Entdeckungen und der daraus ableitbaren Unvollständigkeit des bestehenden Wissens, nicht existiert. In jedem Fall ist klar, dass im Rahmen dieser Studienarbeit nur ein Teilbereich des beabsichtigten Wissens abgebildet werden kann.

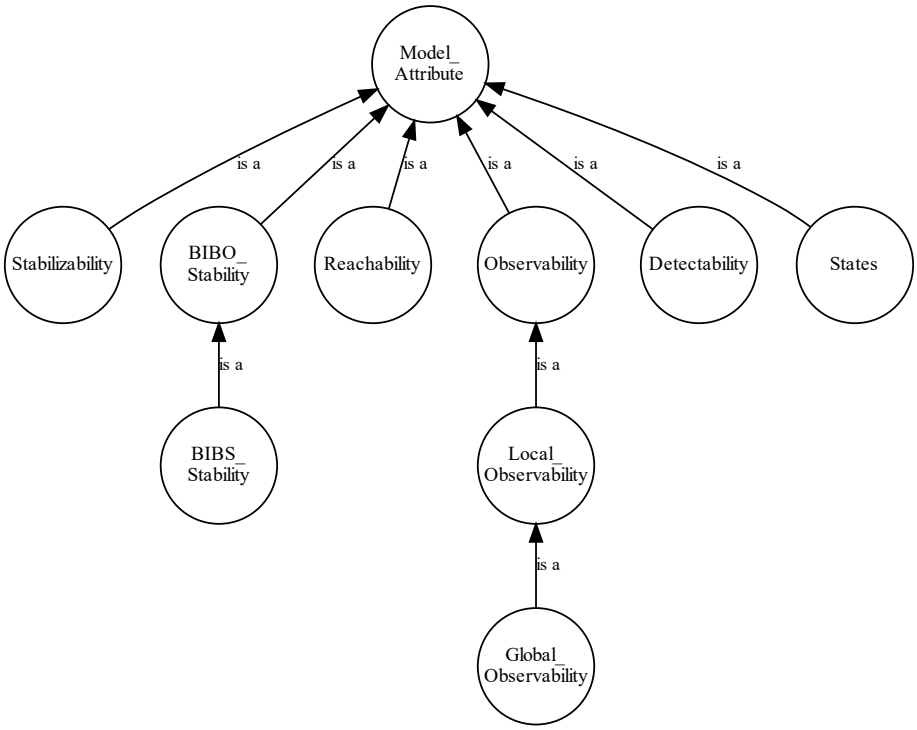


Abbildung 5 – Auschnitt des KS aus der Hauptkategorie *Modelleigenschaften*

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. 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.

Das KS besteht aktuell aus 90 Knoten. Abbildung 5 zeigt ein Ausschnitt des KS.

3.2.1 Aufbau des Klassifikationssystems

Das KS ist ein Semantisches Netz (s. Punkt **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 Modelleigenschaften interpretiert werden.
Die Kategorieknoten im KS werden durch ihren eindeutigen Namen identifiziert (s. Punkt **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 die Knoten der Ursprungs -und Hauptkategorie(-n), entlang des gerichteten Pfades $P(k_i)$ als Modellattribut. Aus diesem Grund wird in der *tag_list* der Metadaten-Datei immer nur die spezifischste Eigenschaften 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 mathematischen 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 Abschnitt 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[2]. Oder einfacher,

[2] vgl. [16, S. 34]

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 durch 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 einen 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 und der Inhalt der Zeile stellt einen Eintrag der Sequenz dar. *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 6 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

```
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 6 – Informationsblöcke des KS

Hilfe des Packages *pyyaml*[3] einliest. Zudem erzeugt es einen gerichteten Graphen des Python-Packages *networkx*[4] und zeichnet den Subgraphen, der nur die Kategorieknoten enthält mit dem Python-Package *nxv*[5] 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 explizit 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 entfernen von Knoten aus dem KS, wird in der Implementierung durch schreiben neuer IB‘s oder entfernen bestehender 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.

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*.

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

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 7 – 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

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*

[6] Modell stammt aus [14] Abschnitt 1.3

2 Model Equations

State Vector and Input Vector:

$$\underline{x} = (x_1\ x_2)^T = (I\ U)^T$$
$$\underline{u} = u_1 = d$$

Model Equations:

$$\dot{x}_1 = -(1-u_1)\frac{1}{L}x_2 + \frac{U_{DC}}{L} \tag{1a}$$
$$\dot{x}_2 = (1-u_1)\frac{1}{C}x_1 - \frac{1}{RC}x_2 \tag{1b}$$

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

Abbildung 8 – Gleichungen aus der Dokumentation zum Hochsetzsteller (Boost-Converter)[6]

und *Beispielhafte Parameterwerte* explizit formuliert. Weitere Sektionen können vom Ersteller des Modells nach eigener, subjektiver Einschätzung der Sinnhaftigkeit hinzugefügt werden. Denkbar wäre, angenommen es handelt sich um ein flaches Modell, z. B. ein Abschnitt der flache Ausgänge angibt. 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.

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 diese 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 bzw. die Eingangsfunktion integriert.
Ein Modell kann also einfach über ein selbst 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 Python-Packages *scipy*[7]) simuliert wird. Die Methode *get_rhs_func*[8] 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*[9] 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* bereits implementiert.
Innerhalb einer Modellklasse sind nur die symbolischen Gleichungen als Code geschrieben. Die Umwandlung in die numerische 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 ein gewöhnliches Differentialgleichungssystem erster Ordnung beschrieben wird, implementiert werden.

3.5 Vorhandene Modelle

Die aktuell im Katalog enthaltenden Modelle sind in Tabelle 2 aufgelistet.

[7] SciPy-Package: https://www.scipy.org/
[8] *rhs* steht für *right hand side*
[9] SymPy-Package: https://www.sympy.org/en/index.html

Nr	Name	Modelldimension	Implementiert?
1	Hochsetzsteller	2	Nein
2	Brockett-Integrator	3	Ja
3	DC-DC Tiefsetzsteller	2	Nein
4	Heisenberg-Flugrad	3	Nein
5	Kapitzas Pendel	2	Ja
6	Lorenz Attraktor	3	Ja
7	MMC[10]	8	Nein
8	N-Fach Integrator	n	Ja
9	PVTOL mit 2 Kräften[11]	6	Ja
10	Rössler Attrkator 1979a	3	Ja

Tabelle 2 – Modelle des Kataloges

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

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

1. Git installieren und das Git-Repositorium(siehe **E.8**) des Kataloges herunterladen um eine lokale Kopie des Kataloges und der Vorlagen(siehe Abschnitt 3.1 Katalogstruktur) zu erhalten.

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

3. Erstellen der *Metadaten-Datei* im Modellordner mithilfe der Vorlage und Umbe-

[10] Modular Multilevel Converter
[11] Planar Vertical Takeoff and Landing [Vehicle] (dt. Senkrechtstarter)

nennung dieser in *metadata.yml*.

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

   Anmerkung 1: Die Vorlage stellt aktuell nur eine einheitliche Notation für gewöhnliche Differentialgleichungssysteme erster Ordnung zur Verfügung. Modelle, deren Modellgleichungen in ein gewöhnliches Differentialgleichungssystem erster Ordnung transformierbar sind, müssen in ein solches transformiert werden.

   Anmerkung 2: Für Modell, deren Modellgleichungen nicht in ein gewöhnliches Differentialgleichungssystem erster Ordnung transformierbar sind, gibt es noch keine festgelegte Notationsvorgabe.

5. Erstellen der *Parameter-Datei* im Implementationsordner mithilfe der Vorlage und ausführen[12] dieser.

6. Die PDF-Datei der Modelldokumentation erzeugen.

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

   Anmerkung: Aktuell können nur Modelle, die durch ein gewöhnliches Differentialgleichungssystem erster Ordnung beschrieben werden, mithilfe der Vorlage implementiert werden.

8. Pull-Request ausführen.

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

Kapitel 4

Zusammenfassung und Ausblick

4.1 Zusammenfassung

In dieser Studienarbeit wurde die Erstellung eines Kataloges von regelungstechnischen Systemmodellen beschrieben. Angefangen wurde in Kapitel 2 mit einer Betrachtung der aktuellen Situation bezüglich der Suche und Implementation von Modellen mit dem Ergebnis, dass diese nicht Optimal ist. Außerdem wurden zwei aktuell vorhandene Modellsammlungen beschrieben.

Danach wurden im Abschnitt 2.3 zuerst Anforderungen formuliert, die der Katalog haben soll und anschließend eine Reihe von Entscheidungen ausgeführt, die der Erfüllung dieser Anforderungen dienen sollen. Im Abschnitt 2.5 wurden Entscheidungen vorgestellt, die aufgetretene Frage- und Problemstellungen beantworten.

In Kapitel 3 wurde der *Katalog dynamischer und regelungstechnischer Modelle* vorgestellt. Zuerst wurde beschrieben aus welchen Elementen sich dieser zusammensetzt und wie diese zusammenhängen. Anschließend wurden die Elemente des Kataloges detaillierter betrachtet. Die letzten zwei Abschnitte lieferten eine Übersicht zu den aktuell vorhandenen Modellen und dem vorgehen zum Hinzufügen neuer Modelle.

Der Katalog enthält aktuell nur einen recht kleinen Umfang an Modellen und ist momentan nur für Modelle, die durch gewöhnliche Differentialgleichungen erster Ordnung beschrieben werden ausgelegt und durchdacht worden. Für diese Modelle wurde eine einheitliche Dokumentationsform sowie eine einfach anwendbare Implementationsweise durch die Python-Klasse *GenericModel* entwickelt. Zudem wurde mit der *Metadaten-Datei* eine Schnittstelle hinzugefügt, die es erlaubt aus dem Katalog eine Datenbank inklusive Suchfunktion zu erstellen.

Es wurde das *Klassifikationssystem* vorgestellt, welches als Wissensrepräsentation für die Teilbereiche der Mathematik und Regelungstheorie, die sich auf Modelle beziehen, gedacht ist. Es enthält die möglichen Eigenschaften der Modelle und stellt deren Beziehung zueinander dar. Das abgebildete Wissen des Klassifikationssystems enthält

die geläufigsten Modelleigenschaften und hat noch viel Erweiterungspotenzial. Es wurde die technische Umsetzung des KS beschrieben, die eine einfache Les- und Editierbarkeit gewährleistet.

4.2 Ausblick

Der Katalog wurde mit der Absicht erstellt die Suche nach Modellen zu vereinfachen und das, hoffentlich, für eine zukünftig große Anzahl an Personen. Um diese Zukunft zu erreichen fehlt es dem Katalog allerdings noch an einigen sinnvollen Elementen. Im folgenden werden einige Ideen formuliert.

Wie schon bei Entscheidung **E.1** aufgeführt ist der Einsatz einer Datenbank sinnvoll, welche die im Katalog enthaltenen Modelle und deren Implementationsstatus, sowie die durch die Metadaten-Datei und deren Verknüpfung mit dem KS zur Verfügung gestellten Informationen enthält. Dafür ist ein Algorithmus sinnvoll der Modelle, Implementationsstatus und Metadaten-Dateien automatisiert ausliest und die Einträge für die Modelleigenschaften und deren Werte zusätzlich auf Sinnhaftigkeit und Korrektheit in Bezug auf die Struktur des KS prüft.
Zur Umsetzung der Datenbank wäre zusätzlich eine Suchfunktion sinnvoll. Datenbank und Suchfunktion würden die Erfüllung der Anforderung **A.1** signifikant verbessern.

Zudem wäre noch ein Entwurf für Darstellungsvorgaben für Modelle sinnvoll, deren Modellgleichungen nicht in ein gewöhnliches Differentialgleichungssystem erster Ordnung überführt werden können z. B. Modelle, deren Modellgleichungen DAEs[1] oder PDEs[2] sind.
Des Weiteren wäre dann für solche Fälle auch der Entwurf einer weiteren Vorlage für deren Implementation zu prüfen und gegebenenfalls umzusetzen.

Natürlich ist eine Erweiterung des Modellumfanges des Kataloges sinnvoll.

Der Entwurf der formalisierten Darstellung sollte durch qualifizierte Personen geprüft werden. Das gilt auch für die aktuelle Umsetzung der Implementierung.
Eine zukünftige Veränderung beziehungsweise Anpassung der entworfenen Python-Klasse *GenericModel* z. B. nach Tests oder zur Funktionserweiterung ist wahrscheinlich. Eine Änderung im Code birgt immer das Potenzial von Fehlern. Für eine schnelle und zuverlässige Prüfung, ob die implementierten Modellklassen nach einer Änderung der Klasse *GenericModel* wie gehabt funktionieren, wäre die Implementation automatisierter Tests(sog. *Unit-Tests*) sinnvoll.

Zum Abschluss werden noch zwei Ideen für das *Klassifikationssystem* ausgeführt.

[1] Differential-Algebraic-Equations (dt. Differential-Algebraische-Gleichungen)
[2] Partial-Differential-Equations (dt. Partielle Differentialgleichungen)

Das KS ist so entworfen worden, das es recht einfach erweiterbar ist. Nun ist es so, dass das KS Wissen, welches innerhalb eines Wissensbereiches verwendet wird über die Verknüpfung von Begriffen abbilden soll und das jeder dieser Begriffe eine in den Wissensbereichen der Mathematik und Regelungs- und Steuerungstheorie gemeinsam akzeptierte Bedeutung hat[3].
Dementsprechend wäre es sinnvoll einen Mechanismus einer geprüften Erweiterung des KS zu schaffen, der dafür sorgt das ein Vorschlag zur Veränderung des KS stets durch qualifizierte Personen geprüft und gegebenenfalls diskutiert wird bevor dieser übernommen wird.

Wie eben schon erwähnt haben die Begriffe im KS eine bestimmte Bedeutung, die durch eine Definition festgelegt ist. Mit wenigen Ausnahmen ist eine Angabe dieser Definition oder einer Referenz zu dieser notwendig, zum Einen um sicher zu stellen, das jeder den Begriff gleich interpretiert und zum Anderen um die Unterschiede zwischen Begriffen ermitteln zu können[4].
Die Referenzen für die Begriffe des KS könnten als weiteres Attribut in die Informationsblöcke(siehe Unterabschnitt 3.2.2) des KS geschrieben werden.
Eine Erweiterung zu dieser Idee wäre, alle für das KS verwendeten Referenzen in einer einzelnen Datei zu sammeln und mit einer Zitier-Signatur zu versehen. Das Konzept solcher Dateien ist nicht neu und findet z. B. bei BibTex-Datenbanken, welche wiederum formatierte Textdateien sind und häufig für wissenschaftliche Arbeiten genutzt werden, Anwendung. Die Zitier-Signaturen könnten in den IB's des KS verwendet werden, was die YAML-Dateien des KS übersichtlicher machen würde.
Die Einbindung des KS in die, weiter oben vorgeschlagene, Datenbank zum Katalog und der zugehörigen Suchfunktion würde eine Ausgabe der Referenzen in einheitlicher Schreibweise ermöglichen.

[3] Diesen Aspekt hat Studer 1998 in seiner Definition der Ontologie in [21] Abschnitt 6.1 formuliert.
[4] Hinweis: Zu den meisten Begriffen im KS ist die entsprechende Referenz bekannt. Diese befinden sich aktuell in einer zusätzlichen, relativ formlosen Datei.

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

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

[13] 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.

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

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

[16] 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.

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

[18] 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.

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

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

[21] 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.

[22] 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.