DISS. ETH NO. 22990

Constraint-Based Methods for Automated Computational Design Synthesis of Solution Spaces

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES of ETH ZURICH

(Dr. sc. ETH Zurich)

presented by

CLEMENS HEINZ WOLFGANG MÜNZER

Diplom-Ingenieur Univ., Technische Universität München

born on 07.05.1985

citizen of Germany

accepted on the recommendation of

Prof. Dr. Kristina Shea
Prof. Dr. Petros Koumoutsakos

2015

Abstract

Computers have the capability to support human designers in a variety of tasks. This includes not only releasing the human designer from routine tasks by design automation but also sparking and supporting innovation and creativity in development processes. In order to support the concept phase effectively, a wide range of possible concepts, which are quantitatively evaluated, should be considered to enable designers to explore the solution space and to make advantageous decisions towards concepts to be considered in consecutive development phases. To enable an automated systematic solution space exploration and evaluation, this research presents different approaches based on a graph-based object-oriented knowledge representation. This representation is combined with first-order logic and Boolean satisfiability as foundation for a generic automated approach for requirement-driven computational design synthesis of solution spaces. To enable the evaluation of the generated solution spaces, a generic approach to automatically translate the generated graph-based product concepts into Bond graph-based simulation models is described. Finally, a method is presented to parametrically optimize the generated concepts using simulated annealing. Here, parameterizations are generated by automatically setting up and solving constraint satisfaction problems and evaluated using the generated simulation models. The methods are validated on the case studies of chemical process engineering, automotive powertrains and 3D-Printer kinematic mechanisms. The main contributions of this research are a continuous and generic approach starting with task definitions and ending with a valid, parameterized product concept, a method which is able to determine if an engineering task is solvable for a given set of synthesis building blocks, and an approach for a generic transformation of the generated product concepts to Bond graph-based simulation models. Thus, this research provides new knowledge in terms of generic transformations between different knowledge representations in order to generate, explore and evaluate large solution spaces with an, until now, unreached expressiveness.

Zusammenfassung

Die fortschreitende Entwicklung von Computern ermöglicht die Unterstützung von Entwicklern bei vielerlei Aufgaben. Diese Unterstützung erlaubt nicht nur, durch Automatisierung, Routineaufgaben des Entwicklers zu übernehmen, sondern kann auch Innovation und Kreativität in Entwicklungsprozessen fördern. Um die Phase des Konzipierens effektiv zu unterstützen muss eine grosse Anzahl von möglichen Konzepten erzeugt und bewertet werden um dem Entwickler eine Beurteilung des Lösungsraumes zu ermöglichen und ihn oder sie damit zu guten Entscheidungen zugunsten eines oder mehrerer Konzepte zu befähigen, die in den folgenden Phasen ausgearbeitet werden. Um eine systematische automatisierte Erschliessung von Lösungsräumen zu ermöglichen präsentiert die vorliegende Arbeit einen dreiteiligen Ansatz der auf einer graphbasierten objektorientierten Wissensrepräsentation aufbaut. Im ersten Teil wird diese Wissensrepräsentation genutzt um mit Hilfe von Prädikatenlogik und boolescher Logik eine generische automatisierte Erzeugung von Konzepttopologien im Lösungsraum zu ermöglichen. Für jede erzeugte Konzepttopologie wird, im zweiten Teil des Ansatzes, ein Constraint Satisfaction Problem erstellt und gelöst um, wenn möglich, auf der Grundlage von Anforderungen und kritischen Betriebspunkten mögliche Parametrisierungen des Konzepts zu erzeugen. Im dritten Teil des Ansatzes wird die optimale Parametrisierung jedes erzeugten Konzeptes mit Hilfe der Optimierungsmethode Simulated Annealing ermittelt. Dazu wird automatisiert für jedes Konzept ein Simulationsmodell in Form eines Bondgraphen generiert und für verschiedene Parametrisierungen ausgewertet. Um den Ansatz zu validieren werden verschiedene Fallstudien betrachtet: ein Beispiel aus der chemischen Verfahrenstechnik, zwei Beispiele zur Entwicklung automobiler Antriebsstränge und ein Beispiel zur Entwicklung von Mechaniken zum Antrieb von Druckköpfen in 3D-Druckern. Der Hauptbeitrag der präsentierten Forschung ist ein kontinuierlicher generischer Ansatz der, ausgehend von einer formal definierten Entwicklungsaufgabe, parametrisierte Produktkonzepte erzeugt und bewertet. In diesem Prozess kann, für einen endlichen Satz an möglichen Bausteinen für die Konzeptsynthese, die Lösbarkeit der Entwicklungsaufgabe untersucht werden. Ausserdem wird ein generischer Ansatz präsentiert der die erzeugten Konzepttopologien in simulierbare Bondgraphen übersetzt. Somit erweitert diese Forschungsarbeit den Stand der Technik mit neuen Ansätzen zu generischen Modell- und Repräsentationstransformationen und ermöglicht so grosse Lösungsräume zu erschliessen und, mit bis dato unerreichtem Informationsgehalt, zu bewerten.

Acknowledgements

This work results from my occupation as a doctoral student in the Virtual Product Development Group at the Institute of Product Development at the Technische Universität München from May 2011 to April 2012 and at the Engineering Design and Computing Laboratory at ETH Zurich from May 2012 to September 2015.

Firstly, I would like to thank my doctoral advisor Prof. Kristina Shea for accepting me as a PhD candidate, for all the good discussions we had, and the time she invested in me. Without her I would not be the researcher I am today. Further, I would like to thank Prof. Petros Koumoutsakos for being my co-examiner and Prof. Boulouchos for chairing my defense.

Looking back at the time of this PhD project I would like to thank all the colleagues at the Institute for Product Development at Technische Universität München and especially my fellow researchers from the Virtual Product Development Group for welcoming me and all their support during my scientific infancy. Especially, I would like to thank Bergen Helms for introducing me to this research field and all the fantastic discourses we had.

At EDAC, my time would not have been that enjoyable without my amazing colleagues. Especially, I want to thank Tino Stanković for accepting me as an apprentice in the dark arts, all his honest and valuable feedback, and countless coffees in numerous places. Similarly, I want to thank Corinna Königseder for her friendship, putting up with me for over three and a half years in one office, all the rock climbing lessons, long discussions and always questioning the sense. Finally, in alphabetical order, there are Allie, Benjamin, Bettina, Eugen, Fritz, Jochen, Jung, Luca, Martin, Merel, Paul, and Tim, who all populated the coffee room, participated in conversations with different intellectual levels, and, therefore, created such a great environment.

I want to thank all my friends in Munich, Zurich and all around the globe. They were all there for me, accepted all my moods and looked past my unsociability in the last months before the submission. I'm especially indebted to Basti for 150k portions of wisdom (still counting), Caro for countless RPs, and Stefan for unbreakable optimism.

Last but definitively not least, I want to express my gratitude to my parents Sabine and Wolf-Heiner Münzer who supported me not only during this thesis but all my life. Also, I want to thank my siblings and their families: Valerie, Flo, Henning, Tanja, and Nathalie for all the encouragement and distraction when needed.

Finally, although doing a PhD is, in itself, a selfish act, I want to dedicate this work to my grandparents Helga and Wolfgang Münzer and Ruth and Heinz Gerhard.

Zurich, January 27th 2016 Clemens Münzer

The following publications are part of the work presented in this thesis:

[1] C. Münzer, B. Helms, and K. Shea. Automated parametric design synthesis using graph grammars and constraint solving. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Chicago, IL, USA, August 2012.

[2] C. Münzer, B. Helms, and K. Shea. Solving Design Tasks in Engineering Using Object-Oriented Graph-Based Representations and Boolean Satisfiability. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Portland, OR, USA, August 2013.

[3] C. Münzer, B. Helms, and K. Shea. Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis. *Journal of Mechanical Design*, 135(110):101001-1-13, 2013.

[4] C. Münzer and K. Shea. A Simulation-based CDS Approach: Automated Generation of Simulation Models Based From Generated Concept Model Graphs. In *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Boston, MA, USA, August 2015.

[5] B. Kruse, C. Münzer, S. Wölkl, A. Canedo and K. Shea. A Model-based Functional Modeling and Library Approach for Mechatronic Systems in SysML. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Chicago, IL, USA, August 2012.

[6] B. Kruse, C. Münzer, S. Wölkl, A. Canedo and K. Shea. Workflow and Modeling Conventions for Function and Product Structure Modeling of Mechatronic Systems in SysML using Libraries. In R. Scheidl and B. Jakoby, editors, *The 13th Mechatronics Forum International Conference Proceedings*, volume 2 of *Advances in mechatronics*, pages 506-513, Linz, Austria, 2012.

Contents

Abstract III

Zusammenfassung V

Acknowledgements VII

1 Introduction 1
1.1 Thesis Structure . . . 6

2 Related Work 9
2.1 Computational Design Synthesis . . . 10
2.1.1 Rule-based Approaches . . . 11
2.1.2 Model-based Approaches . . . 13
2.1.3 Mixed Approaches . . . 13
2.1.4 Conclusion . . . 15
2.2 Approaches Applying Logic Descriptions in Design . . . 15
2.3 Constraint Satisfaction in Design . . . 16
2.4 Simulation-Driven Conceptual Design . . . 17
2.5 Conclusions & Research Questions . . . 19

3 Background 21
3.1 First-order Logic & Boolean Satisfiability . . . 21
3.1.1 First-order Logic . . . 21
3.1.2 SAT and SAT-Solvers . . . 21
3.1.3 Transforming First-order Logic to SAT . . . 22
3.2 Simulation Using Bond Graphs . . . 23
3.3 Constraint Satisfaction . . . 23
3.3.1 Variables . . . 24
3.3.2 Constraints . . . 25
3.3.3 Solver . . . 26
3.4 Optimization with Simulated Annealing . . . 26

4 Method 29
4.1 Metamodel . . . 32
4.1.1 Ports . . . 32
4.1.2 Elements . . . 33
4.1.3 Edges . . . 38
4.2 Topological Synthesis Using First-Order Logic and Boolean Satisfiability 40
4.2.1 First-order Logic Representation of the Metamodel . . . 40
4.2.2 Conversion to SAT and Solving . . . 44
4.2.3 Energy and Signal Conservation check . . . 45
4.3 Variable Assignments Generation Using Constraint Satisfaction . . . 47
4.3.1 Constraint Satisfaction Problem Generation . . . 47
4.3.2 Constraint Satisfaction Problem Solving . . . 49
4.4 Single-Objective Optimization Using Simulated Annealing . . . 50

4.5 Summary 55

5 Implementation **57**
5.1 System overview 57
5.2 Used tools 58
5.2.1 GrGen.Net 4.4 58
5.2.2 Alloy Analyzer 58
5.2.3 Gecode 4.4.0 59
5.2.4 LMS Imagine.Lab Amesim 59
5.3 Metamodel Definition 59
5.4 Application 64
5.4.1 Topological Synthesis 65
5.4.2 Variable Assignment Generation 66
5.4.3 Variable Optimization 67
5.5 Summary 67

6 Case Studies **69**
6.1 Case Study I - Chemical Process Engineering 70
6.1.1 Metamodel 70
6.1.2 Method Application 71
6.1.3 Results 72
6.2 Case Study II - Automotive Powertrains I 74
6.2.1 Metamodel 75
6.2.2 Method Application 75
6.2.3 Results 77
6.3 Case Study III - Automotive Powertrains II 80
6.3.1 Metamodel 80
6.3.2 Method Application 83
6.3.3 Results 84
6.4 Case Study IV - Printhead Drive for 3D Printers 89
6.4.1 Metamodel 89
6.4.2 Method Application 93
6.4.3 Results 94
6.5 Summary 97

7 Discussion **99**
7.1 Contributions 100
7.2 Limitations & Future Work 102

8 Conclusion **109**

References **111**

1 Introduction

Over the last decades, computing power and programming techniques have greatly evolved. These advances enable the support and automation of processes in engineering design [7]. A success story of such design automation can be reported from a different domain, namely VLSI[1] Design [8]. VLSI design deals with the design of microprocessors, combining nowadays billions of transistors on a single die. The growing number of transistors that need to be included to support the growing computing requirements led to the development of automated design approaches, since these numbers could not be handled manually anymore. The success of these efforts can be seen by the fact that Moore's law [9] of 1965, i.e. the biannual duplication of the number of transistors in an integrated circuit, still holds today. In VLSI design almost all steps during development are supported by partly automated compatible software tools.

In engineering design, this level of computational support is still far from being reached. A well established viewpoint of the product development process is presented by Pahl et al. [10]. According to them, the process is subdivided in four consecutive phases: i) planning and task clarification, ii) conceptual design, iii) embodiment design, and iv) detail design. During planning and task clarification, the customer needs, the situation of the market and the company are analyzed. The development task is refined and the requirements list is defined. In the conceptual design phase, different preliminary solutions, i.e. concepts, are generated and evaluated against technical and economic criteria. Having chosen the principal solution, or concept, embodiment design begins. Now, the preliminary form design is carried out and materials are selected. Different layouts are refined until one candidate is chosen, which is developed into the definitive layout including a preliminary part list, production, and assembly documents. In detail design, the product documentation: detail drawings, parts lists and the complete production, assembly and operating instructions, i.e. the product documentation, are produced for the definitive layout.

In later phases of the product development process, computational support is more and more integrated. Modern CAD Tools, like CATIA [11] and Siemens NX [12], enable modeling and dimensioning of parts. The resulting models can be transferred to further analysis tools, e.g. for structural analysis and production by CNC[2] milling machines and rapid prototyping [7].

The support in earlier phases, however, is sparse in engineering practice. To support planning and task clarification, several approaches are available in the research field of requirements engineering [13]. Since this research field is not in the focus of this thesis, a closer look towards this field is omitted. Conceptual design, however, is an inherent and important part of every product development process [10]. Figure 1 shows the so-called "dilemma of product development" [14]: In the early stages of the product life cycle the possibility to influence the costs is high, but, at the same time, in these stages the least is known about the actual costs. Another manifestation of this is the "rule of ten"[15] (Figure 2). According to it, going through the different life cycle stages of a product, the

[1] **V**ery **L**arge **S**cale **I**ntegration
[2] **C**omputerized **N**umerical **C**ontrolled

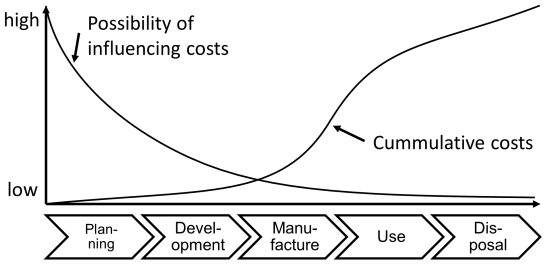


Figure 1: Influence on cost and actual occurrence of cost over the life cycle of a product (reproduced from [14]).

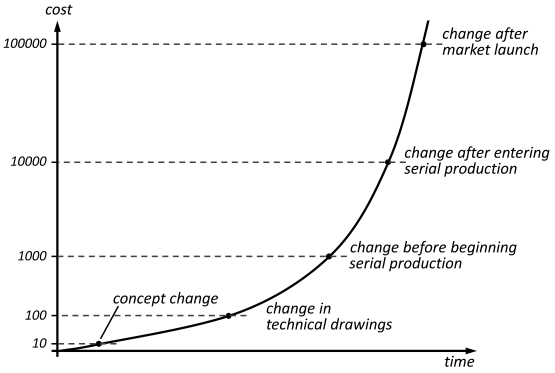


Figure 2: Rule of ten [15] (reproduced from [15]).

cost to fix the same error rises with one order of magnitude for each stage.

The decisions made in the conceptual design phase have a huge impact on the life cycle costs of the product. Although the least amount of quantitative information is known, the most important decisions are made here. To effectively choose which designs are to be taken into embodiment and detail design, a large variety of concepts should be generated and evaluated to fully explore the solution space in order to determine the most promising feasible designs [16]. This need for systematic solution space exploration and the quantitative description of the product requirements are a good precondition for the application of computational design synthesis (CDS), the automated generation of solutions and solution spaces for engineering tasks.

Since the beginning of research in artificial intelligence (AI), methods have been developed that aimed at supporting human designers. The first intention of these approaches was to assist and automate routine and near-routine tasks, e.g. the preparation of technical drawings. Once this field reached a certain level of maturity, research turned to automating non-routine tasks [7]. The research field of CDS, also called formal engineering design synthesis [17], covers such approaches. According to Cagan et al. [18], "computational design synthesis is invoked in situations in which the human designers are often at a loss of what avenues to pursue, or the best method of achieving a solution requires the generation and evaluation of countless alternatives" [18].

By generating these alternatives and, therefore, exploring the solution space, CDS can provide new perspectives since computers are, in contrast to human designers, not biased by known solutions, experience or education [19]. There is also the potential of generating new and innovative designs. Further, encoding knowledge formally can provide a better understanding of the particular design problem following the quote of Knuth [20]:

> "It has often been said that a person doesn't really understand something until he teaches it to someone else. Actually a person doesn't really understand something until he can teach it to a computer [...]"[20]

Case-evidence and a field study gathered and conducted by Thomke and Fujimoto [21] also show the potential of implementing computational techniques in early phases of the

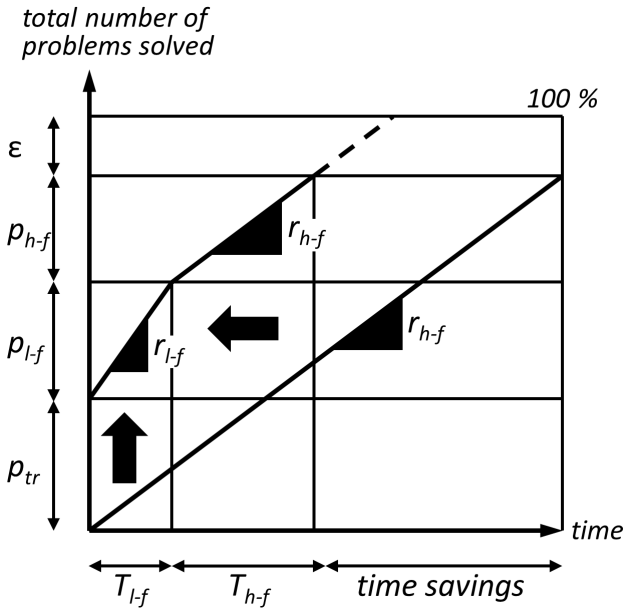


Figure 3: Problem solving rates (r) in product development processes. The right straight represents a conventional product development using high-fidelity methods ($h-f$); the other curve depicts a product development process supported by knowledge transfer (tr) and low-fidelity methods ($l-f$). In order to simplify the figure the trajectories are shown linearly (reproduced from [21]).

product development process that they consider as consecutive problem solving. Figure 3 comprises their findings in two curves depicting the total number of solved problems in a product development process over time. A conventional product development process with a constant problem solving rate ($r_{h\text{-}f}$) using high-fidelity methods (*h-f*), i.e. physical prototypes is depicted as a straight line. The development time can be reduced by transferring knowledge about problems that were solved in previous projects or in other departments of the company (p_{tr}) and the use of low-fidelity methods (*l-f*), i.e. virtual prototypes, digital mock-ups, simulation, etc., right from the start of the development. Although the fidelity of those methods is lower, problems can be identified and solved faster, thus resolving in a higher problem solving rate ($r_{l\text{-}f}$). The same amount of problems can so be solved in less time. The saved time could be used to solve problems (ε) that would remain unsolved in a conventional process to increase product quality.

So, given these advantages and that more computational support in the conceptual phase of product development was already requested in the VDI guideline 2221 [22] in 1993; why is there a hesitation to apply it in engineering practice?

There are several reasons for this fact. Although numerous approaches in CDS have already been presented (cf. Section 2 for details), there are still unresolved issues [19]:

1. One challenge yet to be solved is that being confronted with a huge amount of design alternatives that can result from exploring solution spaces can be overwhelming and overburdening for the user [19]. Most approaches focus on the sheer generation of design alternatives and don't consider advanced methods to evaluate the designs. Due to this, the user is left with a set of solutions but only having limited informa-

tion to rank them or to select a set of solutions for further investigations.

2. Another challenge is that formalizing knowledge is a difficult, tedious and error-prone task that still lacks support [7, 19]. It requires not only a deep understanding of the design task at hand and the engineering problem-solving knowledge, but also a basic understanding of how the selected synthesis method including its operators and algorithms works [23].

3. A related challenge is that different knowledge representations, e.g. statements, sketches, mathematical models, etc., are needed to describe the various aspects of design concepts. Currently, the designer has to manually select the one he or she needs for a particular task. Similar observations can be made regarding representing engineering problem solving knowledge. The importance of including different representations in synthesis and the need to link these representations, either by automated transformations between them or providing a symbolic representation as a point of reference have been identified in literature [7, 18, 24]. This would enable a seamless model, method and tool chain [7, 25].

4. Chakrabarti et al. [19] also state that scaling up current synthesis systems is required to meet the levels of complexity that such systems would most likely face in engineering practice.

5. The last challenge in this list is identified by Eder [26]. He states, that engineering designers usually work within a limited set of design problems where they have sufficient knowledge about the usual processes and solutions. This inhibits the implementation of formal methods, i.e. CDS approaches. Such approaches are only needed if designers are confronted with problems that go beyond their experience.

Although this list is surely not conclusive, addressing all these issues goes beyond the scope of one thesis. Especially the fifth one cannot be resolved by "just" improving the tools and method but also by changing the perception and openness of human designers.

The main research question, this thesis intends to answer is the following:

> How can we computationally generate concept alternatives to a level of completeness so that behavior simulation allows designers to make informed design decisions in conceptual design?

To address the research question, this work presents a new computational design synthesis approach based on a graph-based object-oriented concept modeling framework [27]. Starting with a metamodel, i.e. a set of available building-blocks with defined interfaces and a system boundary that represents the design task as a finite number of inputs that is to be converted into a finite number of outputs, concept topologies are automatically generated and optimized with respect to their parameters. Starting with the building blocks and their inputs and outputs, a first-order logic is generated that describes the solution space of topologies. This first-order logic is then, limited to a maximum number of components in the solution topologies, translated into a Boolean Satisfiability Problem. This, so-called, SAT-Problem is solved, generating the solution space of

topologies. For all the solutions, simulation models are generated using partial simulation models that are predefined for the building blocks. Further, a constraint satisfaction problem (CSP) is generated that covers requirements from the design task and the used building blocks. Solving the CSP, different assignments for the design variables of each topology are created. The most opportune assignment is chosen using an optimization method, i.e. simulated annealing.

Considering the conceptual design phase according to Pahl et al. [10], Figure 4 presents the manual process as described in literature, and the alternative that this thesis proposes. After the essential problems are refined, the metamodel and the system boundary are defined by the designer. Once this task is completed, the method consists of the three automated steps: topological synthesis using first-order logic and Boolean satisfiability, generation of variable assignments using constraint satisfaction, and optimization using simulated annealing, is applied. The generated and evaluated solution space is then transferred to the original process when it is the designers task to evaluate economic criteria, which is not tackled in this work.

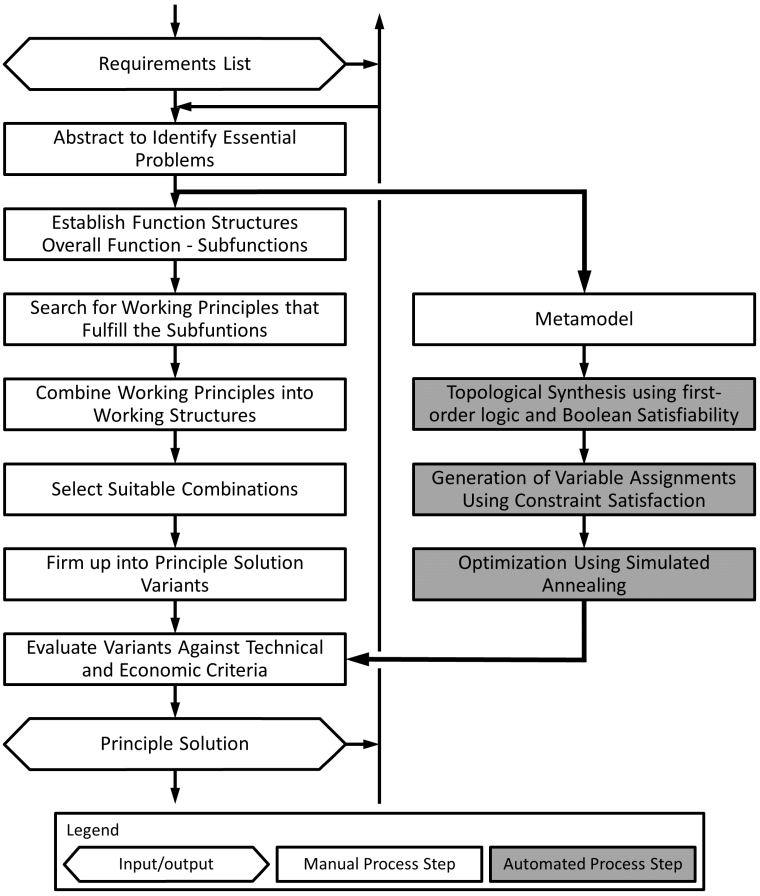


Figure 4: The conceptual design phase reproduced from [10] and the possibilities for automation that this work enables.

In this approach, the knowledge is gathered in the problem specification and the building block set. It is accessed automatically and by the use of simulation, the performance of each generated concept is assessed. Doing this, this work intends to contribute to

the first issue mentioned above that demands the use of advanced evaluation to provide the designer with information to a more informed decision which concept(s) is/are to be followed up in detail design. Providing generic model transformations, translating the metamodel into a first-order logic description, and the generated concepts to CSPs and simulation models, issues two and three are addressed. Since the whole approach is based on a singular knowledge representation, the designer is relieved from manually tackling different representations but still is able to gain the information they provide. The knowledge formalization per se is simplified, since there is only one central location to store the knowledge, i.e. the metamodel. The case studies that are presented in Section 6 address different levels of complexity to contribute to issue four.

The proposed method can also contribute to a shorter development time by i) enabling knowledge transfer in the sense of [21] by offering the possibility to store knowledge in different problem specifications and building blocks and ii) the use of simulation to enable faster problem solving [21]. A better assessment of the different concepts during conceptual design also reduces the probability of errors stemming from this phase which could enable a reduction of cost along the life cycle of the developed product.

Concluding, this thesis' expected contributions are a continuous, generic approach to automatically generate and evaluate topological and variable solution spaces for engineering design problems in the domain of mechanic and mechatronic systems, generic model transformations between the central graph-based object-oriented concept representation and first-order logics, constraint satisfaction problems and simulation models, modeling guidelines for defining simulation models for separate building blocks, and, therefore, an automated method to generate and explore large solution spaces with a "new level of" expressiveness. Further, a software prototype is provided that implements the method and the model transformations. The case studies contribute example metamodels that can serve as basis for future research endeavors and illustrate how the approach can be applied to design problems with different levels of complexity.

1.1 Thesis Structure

Section 2 first gives a general introduction to computational design synthesis before current approaches using different knowledge representations are presented. Further, approaches using logic and constraint satisfaction in design and recent work from the field of simulation-driven design are introduced. Based on this literature review, the focus of this work and refined research questions are elaborated.

Following this, **Section 3** presents the background for the methodologies and approaches that are used in this work: first order logic, Boolean satisfiability, bond graphs, and simulated annealing.

Building on this knowledge, the method is presented in **Section 4**. Here, the framework (Section 4.1) and the three consecutive parts of the method, the topological synthesis (Section 4.2), the generation of variable assignments (Section 4.3), and the optimization

using simulated annealing (Section 4.2) are described in detail.

Section 5 describes the software prototype that was implemented during the work for this thesis. Special focus is given to the definition of the metamodel for the generation and evaluation of concept models and the communication between the different tools utilized in the implementation.

Section 6 presents four different cases studies to show the capabilities of the method. The first case study (Section 6.1) is an example from the domain of chemical process engineering and shows the potentials of the topological synthesis in the domain of process engineering. The second case study (Section 6.2) deals with the generation of variants for automotive powertrains. Here, topological synthesis is compared to a generic approach using an object-oriented graph grammar [27]. The third case study (Section 6.3) offers another viewpoint on automotive powertrains and shows the application of the whole method including integrated simulation. The last case study (Section 6.4) investigates the generation and evaluation for alternative concepts on printhead drives for 3D printers.

Section 7 discusses the presented approach. It is compared to the literature and the contributions are presented. Finally, limitations are analyzed and potential future work is suggested.

In **Section 8** concludes the thesis, summarizing the work and reviewing the contributions considering the challenges identified in this introduction.

2 Related Work

This thesis presents a new method for the generation and evaluation of solution spaces. The intention to support designers in conceptual design is not new and has been tackled by numerous approaches. In order to put this thesis in context with literature and to point out where the expected contributions are placed, this section will introduce the different research fields that are related to this thesis.

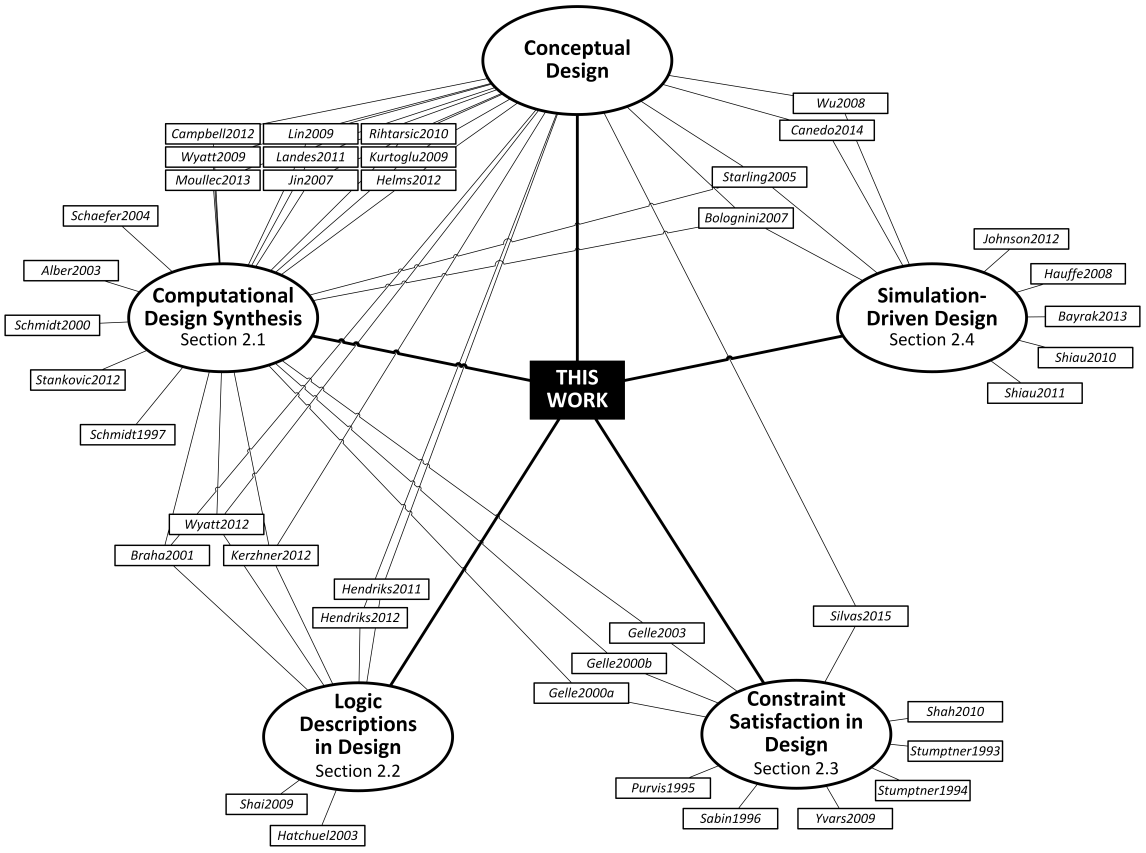


Figure 5: The discussed literature in context of computational synthesis, constraint satisfaction in design, conceptual design, and simulation-driven design

Figure 5 gives an overview about the reviewed literature in this section and its structure. In Section 2.1, the research fields of **computational design synthesis** and configuration are introduced and discussed. Approaches using rule-based, model-based and mixed knowledge representations are presented and compared. Following this, **approaches applying logic descriptions in design** are introduced in Section 2.2. After approaches using **constraint satisfaction in design** are discussed, Section 2.4 reveals details of **simulation-driven design** and related approaches. Section 2.5 draws the **conclusions** from the observations in the related work and refines the research question this thesis intends to answer.

2.1 Computational Design Synthesis

There are two research fields that use AI methods to help designers by creating design alternatives, computational design synthesis and configuration. Mittal and Frayman [28] define a configuration task as follows:

> "Given: (A) a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints (B) some description of the desired configuration; and (C) possibly some criteria for making optimal selections.
>
> Build: One or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements." [28]

Three aspects are pointed out explicitly towards this definition and summarized in [29].

"

1. one cannot design new components during the configuration task;
2. each component is restricted in advance to only be able to "connect" to other certain components in fixed ways (i.e., they can't be modified to get arbitrary connectivity); and that
3. the solution specifies not only the components in the configuration but also how they are related.

"[29]

The first aspect shows to be especially crucial since the level of abstraction in which the components should be defined is not clearly defined [29]. If the components would allow further refinement, i.e. changes in any parameters, there is a chance, that something new is generated. Such incident, the creation of something new can be considered design [29, 17].

Antonsson and Cagan [17] describe synthesis as "many different things, from the physical combination of atoms and molecules into a new molecule, to the combination of existing parts or elements into a larger structure or system"[17]. Following up on this, they characterize engineering design synthesis as "that portion of the engineering design process having to do with the creation of new alternatives and candidate concepts"[17].

Brown [29] sees configuration as an inherent part of the design process. However, configuration aims at supporting designers that know what they need and want to spare the tedious task of combining predefined components, and checking compatibilities and requirement fulfillment. An example for this is the configuration of computer systems as, for instance, realized in the MICON system [30] or the design of paper handling systems as realized in PRIDE [31]. (Computational) Design Synthesis, in contrast, aims at supporting designers at a crossroads not knowing which direction to pursue or when the range

of possible solutions for a task is too wide to be graspable manually [18].

In 2005, Cagan et al. [18] introduced a framework for computational design synthesis. This generic model of computational design synthesis is composed of for major steps that every system that automatically designs must implement:

- Representation
- Generation
- Evaluation
- Guidance

The representation is formulated by the developer of the computational design method to capture the forms or attributes of the design space. Here the knowledge for the design process has to be integrated. Alternative solutions are generated using this representation within the generation task. Each design candidate is evaluated to determine its degree of requirement fulfillment. Based on this calculation the guidance is returned to the search process in order to improve the solution process [18].

Cagan et al. describe the representation as "key challenge in synthesis" [18]. Knowledge for design synthesis can be formalized in several representational foundations [19, 32]. The following sections give an overview of current approaches utilizing rule-based knowledge representations focusing on graph grammars, model-based and mixed knowledge representations.

2.1.1 Rule-based Approaches

Rule-based knowledge representations include knowledge in rules. These rules are, basically, IF-THEN-relations that transform an initial state into a modified one [33].

In 1997 Schmidt and Cagan [34] introduced *GGREADA*[3] (*Schmidt1997*). This approach generates optimal or near-optimal solutions for design tasks using a graph grammar and simulated annealing. They apply this for generating Meccano® set carts using eleven different rules.

Following this work, Schmidt et al. [35] presented four general graph grammar rules to synthesize mechanisms (*Schmidt2000*). These rules are applied to labeled graphs. The approach is shown on the synthesis of epicyclic gear trains.

A framework for engineering design grammars is the Design Compiler 43, which was first presented in 2003 [36] (*Alber2003*). It allows the formalization of knowledge in rules and rule sequences. A key feature of the Design Compiler 43 is the possibility to transform design graphs into analysis models, e.g. FEM, CAD, etc. Recent applications are the design of satellites [37](*Schaefer2004*) and aircraft cabins [38](*Landes2011*).

[3] **G**raph **G**rammar **RE**cursive **A**nnealing **D**esign **A**lgorithm

Starling and Shea [39] present a parallel grammar approach for simulation-driven mechanical design synthesis in 2005 (*Starling2005*). Here, parallel means that the grammar has graph and shape aspects. By applying grammar rules, a product structure is generated which is then translated into a modelica-based simulation. A multi-objective hybrid pattern search controls the synthesis process deciding on the rule applications.

In 2007, Bolognini et al. [40] propose to modify designs by five operators similar to graph transformation rules (*Bolognini2007*). Their method called *CNS-Burst*[4] starts from an initial design and iteratively modifies it until either the maximum number of objective evaluations is exceeded or a predefined termination criteria is reached. The evaluation of the results is simulation-driven, calculates values for design objectives and decides whether the modified graph is to be added to the design archive. The simulation driven aspects of this work and the work of Starling and Shea [39] are further discussed in Section 2.4.

Another example for a rule-based framework is the approach Lin et al. [41] presented in 2009 (*Lin2009*). They synthesize automated gearboxes based on a set of problem specific rules that all involve two basic items: gears and shafts.

Another rule-based approach is the use of graph grammars to automatically synthesize electromechanical designs by Kurtoglu and Campbell [42], also presented in 2009 (*Kurtoglu2009*). Here, 189 rules were extracted during an empirical analysis of 23 products.

Whereas this approach is applying a breadth-first search to generate as many design alternatives as possible, Campbell et. al [43] introduced a stochastic tree-search algorithm that only generates a small subset of solutions that intends to closely match the user's goal (*Campbell2012*).

Stanković et al. [44] introduced formal models of technical processes and technical process synthesis (*Stankovic2012*). All relevant knowledge is stored in a library of production rules that is created for specific problems. The formal models are validated using a stiffened panel assembly.

Rule-based approaches are a versatile way to encapsulate engineering design knowledge. However, their concept representations suffer from a weak expressiveness. The rules are often tailored for specific applications or classes of problems, which makes them hard to be transferred to new classes of problems. The rule application is often controlled by complex search algorithms. Another disadvantage, that partly arises from these facts, is that defining rules, and rule application sequences or algorithms is difficult for designers and not sufficiently supported [7, 19].

[4] **C**onnected-**N**ode **S**ystem

2.1.2 Model-based Approaches

Model-based knowledge representations capture knowledge in model libraries that can include inheritance. The model libraries and models can be re-used in different instances and combinations. Model-based reasoning methods enable the generation of models to solve specific tasks [33].

An example for an approach using this kind of knowledge representation is the generation of concept variants by the approach of Wyatt et al. [46, 47] that was first presented in 2009 (*Wyatt2009/12*). Here, engineering knowledge is encapsulated within a schema. Available components and connections are organized in hierarchical structures and the knowledge about their appearance and connectivity are represented using constraints. The actual synthesis is an adaption of an initial design by four generic operators.

In 2010 Rihtaršič et al. [48] published a tool called *Sophy*[5] (*Rihtarsic2010*). The foundation of this work is a library of physical laws that are represented in so called basic schemata models. The method creates chains of such basic schemata models and interconnects them to create a model that matches a given set of input and output variables.

Moullec et al. [49] present an approach for system architecture generation and performance assessment using Bayesian networks (*Moullec2013*). They propose a way to capture expert knowledge in a Bayesian network and an algorithm to generate all possible product architectures also considering uncertainty. However, the performance assessments presented in this paper are only limited to basic calculations rather than simulations.

In comparison to the rule-based approaches, the model-based approaches provide more expressive concept representations but currently only use simple search methods. Increasing the search space, the computational effort grows. Due to this, there is a need for more sophisticated search methods including design heuristics combined with a model-based representation.

2.1.3 Mixed Approaches

More sophisticated search methods are used in rule-based approaches. So, several approaches [50, 27] resulted that combine aspects of both, rule- and model-based knowledge representations.

The first mentioned is HiCED[6], a computational design synthesis approach by Jin and Li (2007) [50] (*Jin2007*). This method covers a rule-based functional decomposition process by seven different rules and the coevolutionary approach that evolves the functional model and maps means to functions using frame-based represented knowledge.

In 2012, Helms and Shea [27] introduce an object-oriented graph grammar approach for computational design synthesis (*Helms2012*). Since this approach is the predecessor

[5] **S**ynthesis **o**f **phy**sical laws
[6] **H**ierarchical **C**o**E**volutionary **D**esign

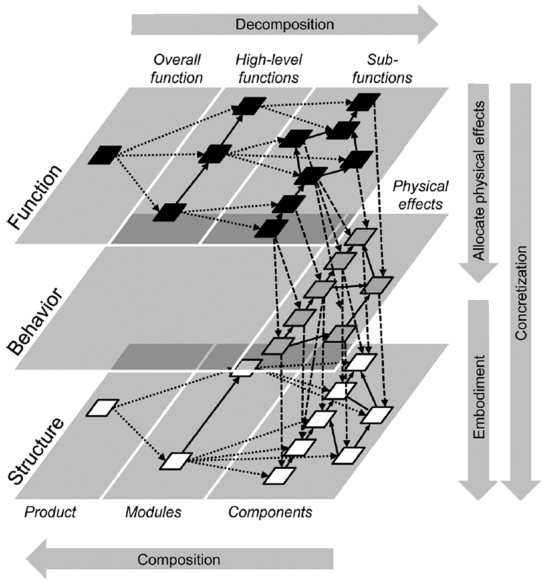


Figure 6: Graph-based product model using function, behavior, and structure levels and design synthesis process steps; adapted from [27]

of this thesis and in Section 6.2 results will be compared, it will be described in more detail. The work is based on a product model using function, behavior, and structure levels (FBS). As depicted in Fig. 6, the approach starts with an overall function that describes the purpose of the product. Manually, this overall function is decomposed into so-called high-level functions that embody main functionalities of the product. Based on the input- and output-flows (ports) of these high-level functions, they are decomposed into chains of subfunctions that are based on the Reconciled Functional Basis [51]. To these subfunctions, behaviors, i.e. physical-effects, are assigned by port-matching [52]. The same is done to map behaviors to components. Finally, components that are not allowed to appear multiple times in the product model are merged. Through this, the isolated FBS-models of the different high-level functions are combined in one product model. The synthesis steps from high-level functions to merging structure components are automatically realized by a set of only four generic graph grammar rules that use the hierarchy of inheritance in which all available components are defined and the components' attributes. Further details can be found in [27, 53].

These approaches show a possible way to combine the advantages of both model-based and rule-based approaches. Since Jin and Li (2007) [50] only use the rule-based approach to generate an initial functional model and continue strictly model-based it suffers from the model-based shortcomings mentioned above. The approach of Helms and Shea [27] shows a more promising way by accessing the knowledge in the model through generic graph rules. However, this approach also has disadvantages: The limited number of generic rules and the sequence of rule applications allows only the generation of simple product topologies for each high-level function, i.e. chains, or structures with at most one branch point. Only merging the structure topologies of the high-level function allows more complex structures. The disadvantage of not allowing highly intertwined topologies to realize high-level functions and the strictly top-down process prevents a wider solution

space exploration. Another limiting factor is the use of random walk search to guide the rule application and, therefore, the concept generation.

2.1.4 Conclusion

Throughout this section, approaches in computational design synthesis using different kinds of knowledge representations research fields are presented. Most approaches ([43, 41, 46, 47, 48, 38, 42]) considered computational design synthesis for conceptual design (cf. Figure 5). This is in line with the characterization of Cagan et al. [18], that "computational design synthesis is invoked in situations in which the human designers are often at a loss of what avenues to pursue, or the best method of achieving a solution requires the generation and evaluation of countless alternatives" [18]. Such situations are common in conceptual design, especially when new products are to be developed. Other approaches in CDS either aim at supporting embodiment design [37], are located even before conceptual design, i.e. synthesize processes [44], or just show their methods on small examples without a direct relation to the product development process [36, 34, 35].

Apart from the disadvantages already mentioned in the different subsection and although the presented approaches constitute sufficient ways to generate solution spaces, most of them lack a sufficient evaluation of the generated concepts. Only the approaches of Starling and Shea [39] and Bolognini et al. [40] include aspects of simulation-driven design which will be further discussed in Section 2.4.

2.2 Approaches Applying Logic Descriptions in Design

Already in 1969, Simon [54] stated "[. . .] that the requirements of design can be met fully by a modest adaptation of ordinary declarative logic"[54]. However, back then, no system was available to actually tackle design tasks [54]. Today, different approaches using logic exist.

An example is the 2001 approach of Braha [55] incorporating Boolean logic (*Braha2001*). Here, a design problem is encoded as a Boolean satisfiability problem and a SAT-solver (see Section 3.1) searches for a finite sequence of production rules. Executing this sequence, the description of a product design that fulfills the initial requirements results. The Boolean variables relate to the existence of structural and functional attributes. The knowledge that is needed to solve the design task is encoded in a set of structural attributes, functional attributes, and production rules that set the two kinds of attributes into relation.

In 2003, C-K theory was introduced [56] (*Hatchuel2003*). Logic is the foundation of this design theory. This theory separates in two spaces: the "space of knowledge" (K) and the "space of concepts" (C). K is a space of propositions that have a logical status (are either true or false). In C there are propositions that have no logical status. Shai et al. [57] used this theory combined with infused design to discover the face force variable in

determinate trusses (*Shai2009*).

Hendriks and Kazakci [58, 59] extend and generalize the method of Semantic Tableaux to Design Tableaux (*Hendriks2011/12*). It is related to CK-Theory and offers a possibility to reason if a generated design concept fulfills a specific design task. The approach shows the capabilities of description logics to evaluate the validity of product concepts. The origin of the concepts, however, is not described. Nevertheless, they show the capabilities of logic in design.

Apart from C-K theory, Wyatt et al. [47] give an outlook for the use of logic in general and SAT in particular to generate product architectures from their design schema. However, they identify certain constructs from their method as not translatable. Additionally, the derivation of architectures from existing product architectures, in their attempt to transfer the system to SAT, is not possible. However, this derivation is a crucial step in their approach.

Another approach to conceptual design using logic descriptions is the work of Kerzhner [60] (*Kerzhner2012*). Here, mixed linear integer programming is used to formalize solution spaces for product architectures and parameterizations. Constructs of the system modeling language SysML [61] are used and, where needed, extended to define architecture selection decisions that are then translated into mixed linear integer programming. There, it is solved and possibly optimized.

Regarding Figure 5, Hendriks and Kazakci [58, 59] use logic reasoning to in conceptual design, but (until now) only apply manual reasoning without any computational support (although they claim such could be possible). Linking Boolean satisfiability and computational design synthesis is only discussed but not implemented in the work of Wyatt et al. [47]. Braha [55] only uses Boolean logics to control a rule-based system that is designed for conceptual design but does not consider the direct use of logic in synthesis. The approach of Kerzhner [60] uses mixed linear integer programming but also lacks a sufficient evaluation of the generated concepts. This approach shows the potential of logical descriptions, but the progress that has been made in research on algorithms to solve Boolean satisfiability problems, the superordinate problem of logical descriptions [55], remains to be investigated in direct application to design tasks.

2.3 Constraint Satisfaction in Design

Another way to implement computational support in design tasks, is to describe the tasks as constraint satisfaction problems (CSP) and use constraint solving algorithms to generate solutions. CSPs are composed of three parts: a set of variables, a set of domains, with a domain, i.e. the possible values, for each variable, and a set of constraints, where the variables are linked with each other. Having defined a CSP, variable assignments that fulfill all constraints can be generated using search methods (cf. Section 3.3). Especially for configuration tasks as described in Section 2.1, constraint programming, i.e. encoding variable relations in constraints and constraint satisfaction problems, is "the method of choice" [62].

There are several examples for the application of constraint satisfaction for configuration tasks. Stumptner and Haslböck [63] developed COCOS[7] which allows defining and solving configuration tasks based on a generative constraint formalism [64] (*Stumptner1994/93*). Purvis and Pu [65] introduce an adaptation approach using constraint satisfaction in the field of case-based reasoning and test it on configuration tasks and assembly sequence generation (*Purvis1995*). Sabin and Freuder [66] realize configuration tasks using composite constraint satisfaction (*Sabin1996*). Here, values for certain variables can be constraints themselves. If such a value is assigned, the constraint satisfaction problem is dynamically updated. An application to the configuration of automotive powertrains is presented in [67] (*Silvas2015*).

The approaches presented until now are limited to discrete variable definitions. Adding continuous, or numerical, ranges for variables, Yvars et al. [68] optimize mechanical systems using constraints satisfaction (*Yvars2009*). Additionally, Gelle et al. [69, 70] developed constraint satisfaction methods to support mainly structural engineering by generating solution spaces for engineering problems, i.e. single story steel buildings (*Gelle2000a/b*). This research led to a method for solving mixed, i.e. including discrete and numerical variables, and conditional, i.e. the existence of variables depends on the values of other variables, constraint satisfaction problems [71] (*Gelle2003*). Using numeric variables and constraints, Shah et al. [72] combined SysML and constraint-based reasoning for component sizing in hydraulic systems (*Shah2010*). Due to including numerical values for variables, these approaches go beyond the notion of configuration tasks and towards computational design synthesis, since components can change throughout the configuration of solutions.

These approaches show, that CSPs can be successfully applied to configuration and design. However, the expressiveness of the representations is often limited and the decisions made during solving the CSP are only based on basic calculations. None of the approaches tackles the analysis of the design's behavior using simulation or other advanced methods. This leads to the notion of simulation-driven design. Different approaches supporting this paradigm will be presented in the following section.

2.4 Simulation-Driven Conceptual Design

"Simulation-driven design can be defined as a design process where decisions related to the behavior and the performance of the design in all major phases of the process are significantly supported by computer-based product modeling and simulation" [73]. To support this methodology, different approaches have been presented.

In [74, 75] Shiau et al. present an automated approach to find an optimal hybrid-electric vehicle design and allocation in order to achieve minimum life cycle costs, petroleum consumption and greenhouse gas emissions (*Shiau2010/11*). The powertrain topologies

[7] **CO**nfiguration through **CO**nstraint **S**atisfaction

investigated are limited to the topology used in the Toyota Prius and a comparable conventional vehicle, i.e. a Honda Accord. The results are generated by optimizing the parameters in the powertrains and simulating them using the Powertrain Systems Analysis Toolkit.

A similar approach was published in [76] (*Hauffe2008*). Here, the parameters in a basic parallel hybrid powertrain topology are changed to investigate the impact of battery weight and charging patterns on fuel consumption, CO_2 emissions and costs. Similar to [74, 75] different powertrain topologies are not considered.

Another example using the case study of automotive powertrains is shown in [77] (*Canedo2014*). Here, the Functional Modeling Compiler is used to generate topologies to a functional model which are then connected to simulation models. However in the case-study, the simulation models for the different powertrains are hard-coded and then linked to the topologies.

The approach of Starling and Shea [39] also includes aspects of simulation driven design. As mentioned before, they introduce a parallel grammar for simulation-driven mechanical design synthesis. By applying grammar rules, a product structure is generated which is then translated into a Modelica-based simulation. To simulate different designs, precompiled simulation models are used, that have to be created beforehand for every design family that is to be investigated. During the synthesis, only the variables in this precompiled models are changed.

In [78] Bayrak et al. present an automated approach to generate driving modes for hybrid-electric vehicle architectures (*Bayrak2013*). The different driving modes are to be realized using different connection schemes of two planetary drives, two electric engines, and one internal combustion engine. To evaluate the driving modes, a bond graph model for the basic configuration is created and then the computer iterates over all the different connection possibilities. In the end the two modes are selected that are, in combination, able to provide the minimum energy consumption for the vehicle. Whereas this approach verifies the suitability of bond graphs to simulate multi-domain systems, the approach is case-specific for powertrains that involve the components mentioned above.

Also, Wu et al. [79] present an approach in simulation driven design (*Wu2008*). They aim at supporting designers with a tool to automatically create bond graph-based simulation models during manual concept modeling of dynamic systems. For modeling the concepts a new notion, so-called conceptual dynamics graphs, is introduced. The user connects predefined components from a repository that are associated with generic simulation models. The tool assembles such generic simulation models using various types of connectors while the user is modeling and, in case of doubt, includes the user in decisions over compatibility. When the concept is finished, an automated optimization of design variables using genetic algorithms is carried out.

Another approach supporting manual product design is the integration of SysML and Modelica introduced by Johnson et al. [80] (*Johnson2012*). Their work defines an ap-

proach to model system dynamics in SysML by integrating Modelica code for specific components in SysML's Block Definition Diagram (BDD). The blocks defined in such BDD can then be used in parametric diagrams to manually build up systems that are then translated into Modelica code using a triple graph grammar.

The presented approaches can be distributed into three different categories:

1. Approaches using one [74, 75, 76] or several [77, 39] hard-coded or precomiled simulation models.
2. Approaches alternating specific connections in simulation models to determine and optimize different modes of use [78].
3. Approaches supporting human designers in manual concept design [80, 79].

The only approach adapting different architectures into simulation models is the approach of Bologini et al. [40]. However, their application in MEMS[8] is highly problem specific and can be classified as structural topology optimization [81]. Thus, their method cannot be accounted for a generic approach towards using simulation for evaluation.

2.5 Conclusions & Research Questions

Concluding from the different approaches that were presented in this section, this work intends to contribute by providing a seamless approach spanning all five research fields, i.e. the fully automated generation and evaluation of solution spaces for conceptual design. In order to achieve this, engineering knowledge has to be encapsulated in formal, i.e. computationally accessible form. Logical descriptions can be used to formalize design tasks, but the manual formulation of abstract logical statements is tedious and error prone. Formulating engineering knowledge in rule-based knowledge representations raises the conflict as to where to include the knowledge: into the building block types or into the rules. Basing on more expressive building block types enables a synthesis with a small number of rules [27] whereas a huge set of rules is needed to handle more basic building block types [37, 38]. Pushing the expressiveness so far, that no rules are needed anymore would resolve the conflict. Model-based approaches show potential but, as mentioned above, lack of advanced search methods when it comes to synthesizing solutions. These observations lead to the refined research question one:

> 1. How can we formally encapsulate a task specification and engineering knowledge without using rules for conceptual design to allow the generation and automated behavioral simulation of the solution space using advanced search methods?

Given the different aspects of engineering design, there is no one model or language that can include all knowledge and information that is needed to solve engineering design tasks [80]. Almost every approach presented in this chapter utilizes individual formalizations. Different approaches have shown potential in different tasks during design, e.g.

[8] **M**icro**E**lectro**M**echanical **S**ystems

constraint satisfaction problems are suitable for sizing components, and simulation models give an outlook to the real world behavior of a system. In order to make use of suitable representations, refined research question two results:

> 2. How can we implement and execute model transformations to support automating conceptual design?

As mentioned in Section 1, an almost complete design automation has already been achieved in VLSI design [8]. An advantage in VLSI is that the different components are highly independent from each other and only communicate at defined interfaces. Designing a new chip is, therefore, based on combining generic components that are developed separately from the overall system [8]. Unfortunately, this cannot be directly applied to mechanical engineering. In mechanical or mechatronic systems, the involved components influence each other and show different behavior dependent on their deployment. Until now, simulation-based support for conceptual design relies on hard-coded simulation models for different overall concepts. Kerzhner [60] points out, that assigning and maintaining a well defined causality in simulation models created from generated concepts is a "significant problem". Following this, refined research question three is defined:

> 3. How can we enable a mapping between abstract components and simulation models describing their behavior for computational design synthesis?

This thesis intends to provide answers to all the three research questions and by that answer the greater research question raised in the end of Section 1. A model-based knowledge representation is used to encapsulate the engineering knowledge and the design task itself. First-order logic and Boolean satisfiability are used to generate alternative concept topologies. Solving constraint satisfaction problems generated from these topologies ensure the feasibility of the topologies and generate variable assignments out of which an opportune assignment is chosen employing simulated annealing. The following section will give the background for the different formalisms used in this work, before Section 4 describes the method.

3 Background

In this chapter, the background for the different methodologies and concepts is presented. Section 3.1 gives the background on first order logic and Boolean Satisfiability. In Section 3.3 characteristics, constituents and solving techniques for constraint satisfaction problems are described before Section 3.2 introduces bond graph modeling and simulation. Finally, Section 3.4 gives a short introduction to optimization using simulated annealing.

3.1 First-order Logic & Boolean Satisfiability

3.1.1 First-order Logic

To explain the main components of first-order logic, the following, well known, example is used [82]:

1. Socrates is human.
2. All humans are mortal.
3. So, Socrates is mortal, too.

The possibility of being human is a parametric expression *is_human*(x), which is true or false dependent on the argument x. For the individual *x = Socrates* the expression is true, but for different arguments, e.g. *x = Table*, it is wrong. An atomic statement whose truth value depends on one or more arguments is called *predicate*. The second statement is a quantitative expression about individuals. It says that *all* humans are mortal. Written in logical terms this can be read as follows: for all x is valid: If *is_human*(x) is true, *mortal*(x) has to be true as well.

Enhancing the propositional calculus, a logical combination of elementary statements that are either true or false (atoms), by predicates with one or more arguments and the possibility to quantify statements leads to first-order logic. Using this logic, the example from above can be written as follows:

1. *is_human*(Socrates)
2. $\forall x(\textit{is\_human}(x) \rightarrow \textit{mortal}(x))$
3. $\models$ *mortal*(Socrates)

3.1.2 SAT and SAT-Solvers

SAT (Boolean Satisfiability Problem) describes decision problems of propositional logics. In fact, the SAT problem is the question whether a propositional formula is satisfiable. That means that there is at least one assignment for the variables so that the overall formula returns true. Since it only covers propositional formulas, SAT problems have only binary variables (*true* or *false*).

The SAT problem is NP-complete[9] [82]. SAT solvers are software tools that try to prove the solvability of a Boolean formula by finding an assignment for each variable that returns *true* for the complete formula. The clausal conjunctive normal form (CNF) [82] is the input format for such solver. As shown in Eq. 1, a CNF (right side) is a set of clauses (in parentheses) interpreted as conjunctions and a clause is a set of literals $\{C_1, C_2, C_3\}$ interpreted as disjunction.

$$(C_1 \vee C_2) \wedge (C_1 \vee C_3) \Rightarrow (C_1, C_2), (C_1, C_3) \tag{1}$$

There are two different SAT-solver categories: solvers working with modern variants of the DPLL[10]-algorithm, a backtracking search algorithm, and solvers using stochastic local search [83]. In this work conflict-driven solvers are used. These solvers extend the DPLL-algorithms to raise efficiency and applicability to large SAT problems [84, 33].

The backtracking algorithm selects a variable in the propositional formula and sets it either *true* or *false*. Then, the satisfiability of the simplified formula is recursively checked. If it is not satisfiable, the opposite truth value is assigned and the same recursive check is carried out. Doing this, the original problem is split into two sub-problems. These sub-problems are simpler, because all clauses that are *true* under the assumption can be removed as well as all literals that are *false*. DPLL enhances backtracking by unit propagation, which deals with unit clauses. Unit clauses are clauses that only contain one literal. The assignment of the variable is set to the truth value, such that the unit clause returns *true*. Another enhancement is pure literal elimination. Pure literals are propositional variables where either all appearances are negated or not. For these variables the truth value is assigned which makes all clauses that contain this variable *true* [33]. The conflict-driven solvers offer further extensions, i.e. efficient conflict analysis, clause learning, backjumping (non-chronological backtracking), activity heuristics, constraint removal, and random restarts. For further information on these solving strategies please refer to [84, 33].

It is also possible to iterate through all possible solutions. To achieve this, the variable assignment that is found as a solution is excluded by negating it in another clause. Solving the adapted problem will generate a new result (if existent). By following this strategy, all solutions to a SAT problem can be generated.

3.1.3 Transforming First-order Logic to SAT

To transform first-order logics into Boolean satisfiability problems, this work uses the approach presented by Torlak [85]. This transformation generates a Boolean satisfiability problem that is able to generate models to the first-order logic, given a finite world. That means, that the set of possible assignments for arguments must be constrained. The transformation is carried out by representing each predicate as a matrix of Boolean variables over the finite space. Quantifiers and logic combinations of the predicates are transferred by specified matrix operations. The result is a Boolean satisfiability problem that can be solved by a SAT-Solver. A Boolean solution, if existent, can then be traced back to the

[9] **N**ondeterministic **P**olynomial time
[10] **D**avis-**P**utnam-**L**ogeman-**L**oveland

first-order logic, generating a model to the logic. For further details on the transformation, the works by Jackson [86] and Torlak [85] are recommended.

3.2 Simulation Using Bond Graphs

This work uses bond graph modeling [87, 88], a graph-based, multi-domain modeling paradigm that uses declarative modeling. Bond graphs are based on the conservation of energy. Along a bond (half-arrow), power P is transmitted between different elements modeled by two so-called power variables: effort e and flow f. The amount of power at each time step t can be calculated as shown in Eq. 2.

$$P(t) = e(t) \cdot f(t) \tag{2}$$

Apart from the bonds that are illustrated by half-arrows, bond graphs are built from nine different basic elements. There are two kinds of energy conserving junctions: a 1-junction where the flow at each bond is constant and a 0-junction where the effort at each bond is constant. Additionally, sources of flow (S_f) and sources of effort (S_e) can be modeled. The remaining possible constituents are inertias (I), capacitors (C), resistors (R), transformers (TF), and gyrators (GY). In order to influence their characteristics with signal-flows modulated versions of inertias, capacitors, resistors, transformers, and gyrators are used.

In order to get computable equations from a bond graph, the so-called causality in the bond graph has to be well defined. Causality refers to the fact that on every bond, the element on one side defines the effort variable and the element on the other side defines the flow variable. The side of the bond that defines the flow is marked with a dash, the so-called causal stroke. Sources of flow (S_f) and inertias (I) define the flow variable coercively. Sources of effort S_e) and capacitors (C) always define the effort variable. Figure 7 shows the different elements and their possible causalities towards a junction (J). After setting up the general layout of a bond graph, the causality is forwarded from the elements with fixed causality over the junctions to the rest of the elements. If invalid causal patterns, i.e. causal conflicts result the model must be changed to enable a valid causality. For further information about the bond graph methodology please refer to [87, 88].

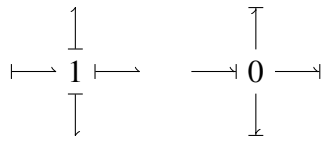
3.3 Constraint Satisfaction

After first approaches appeared in computer science literature in the 1960s, constraint satisfaction became of interest for the research field of artificial intelligence in the 1970s [62]. A set of objects, where each object's condition (example: a switch that is enabled or disabled) has to satisfy a set of constraints is called a constraint satisfaction problem (CSP).

In general, a CSP is composed of three constituents [62]:

1. an n-tuple of variables $X = \langle x_1, x_2, \ldots, x_n \rangle$
2. an n-tuple of domains $D = \langle D_1, D_2, \ldots, D_n \rangle$ with $x_i \in D_i$

Junctions:



Sources:

$S_f \longmapsto J \quad S_e \longrightarrow\!\shortmid J$

Inertia:

$J \longrightarrow\!\shortmid I$

Capacitor:

$J \longmapsto C$

Resistor:

$J \longrightarrow\!\shortmid R \quad J \longmapsto R$

Transformer:

$J \longrightarrow\!\shortmid TF \longrightarrow\!\shortmid J \quad J \longmapsto TF \longmapsto J$

Gyrator:

$J \longrightarrow\!\shortmid GY \longmapsto J \quad J \longmapsto GY \longrightarrow\!\shortmid J$

Figure 7: Bond graph constituents and their causalities

3. a t-tuple of constraints $C = \langle C_1, C_2, \ldots, C_t \rangle$

The following sections present variables and domains, as well as, different types of constraints and solving algorithms.

3.3.1 Variables

Variables, in contrast to constants, have no definite value within the constraint context. For each variable, a domain is defined which incorporates all possible values of the variable are defined.

There are two different kinds of variables: numeric and discrete. The domain for a discrete variable is defined with all explicit values that can be assigned. As an example, in Eq. 3 the domain for the variable *type* is shown. This variable can have two assignments, either *"spur gear"* or *"planetary gear"*. As can be seen in Eq. 4, the domain for numeric variables is defined by the maximum and minimum value that can be assigned. So, this particular domain could also be formulated as $0.6 \leq i \leq 5.0$. Domains containing discrete values are defined using curly braces ($\{\}$) whereas for numeric domains square brackets ($[]$) are used.

$$D\,(type) = \{\textit{"spur gear"}, \textit{"planetary gear"}\} \tag{3}$$

$$D\left(i\right) = [0.6, 5.0] \tag{4}$$

3.3.2 Constraints

Constraints describe mandatory relations between variables. There are three different kinds of constraints as described within the work of [71]:

- numeric constraints
- discrete constraints
- mixed constraints

Numeric constraints are mathematical equations or inequalities and only relate to numeric parameters. An example of such a constraint would be that variable i_2 has to be bigger than the variable i_3. This example is taken from a gearbox where the transmission ratio of the second gear has to be bigger than the transmission ratio of the third gear. These numeric constraints are written as equations or inequalities [71]. The constraint in this example would be defined as shown in Eq. 5.

$$i_2 > i_3 \tag{5}$$

Discrete constraints handle discrete variables. These constraints' representations are started by a C. After that, the involved variables are given in parentheses. On the right side of the :=-sign follow allowed combinations of variable assignments inside curly braces [71]. An example is the correlation between type and supplier of, for instance, a gearbox as shown in Eq. 6.

$$C\left(\textit{supplier}, \textit{type}\right) := \{(\textit{"ZF"}, \textit{"planetary gear"}), (\textit{"Getrag"}, \textit{"spur gear"})\} \tag{6}$$

Mixed constraints, the third constraint type, combine numeric and discrete variables. The illustration is similar to that of discrete constraints. First, the involved variables are referenced and, after that, allowed combinations are defined. In the numerical variables not only explicit values can be set, but also their domains can be restricted or formulas can be given connecting the numeric variables [71]. Three examples are given showing the capabilities of these constraints. Eq. 7 shows an example, the connection between the gearing *type* and efficiency *eta* of a gearbox.

$$C\left(\textit{type}, \textit{eta}\right) := \{(\textit{"planetary gear"}, 0.95), (\textit{"spur gear"}, 0.98)\} \tag{7}$$

The second example is shown in Eq.8. It connects the variable *type* to the domain of the variable n representing the possible range of power input.

$$C\left(\textit{type}, n\right) := \{(\textit{"planetary gear"}, [900, 7000]), (\textit{"spur gear"}, [800, 4800])\} \tag{8}$$

The third example combines the geometrical *shape* of, for instance, a tank (*"hemispherical"* or *"cylindrical"*) with its shape parameters volume V, diameter d and height h.

$$C\left(\textit{shape}, V, d, h\right) = \left\{ \left(\textit{"hemispherical"}, V = \frac{1}{12}\pi d^3 + \frac{1}{4}\pi d^2\left(h - \frac{d}{2}\right)\right), \left(\textit{"cylindrical"}, V = \frac{1}{4}\pi d^2 h\right) \right\} \tag{9}$$

Besides these three types, there is another constraint type: global constraints. They capture relations between a non-fixed number of variables. An example for this is the constraint $\texttt{alldifferent}(x_1, \ldots, x_n)$ meaning that the values assigned to the variables $x_1, \ldots, x_n$ have to be pairwise different [62]. Typically, these constraints can also be realized by a set of simpler constraints. The advantage of global constraints is twofold. First, the clarity during constraint programming is improved and second, these constraints can be solved faster than a set of simpler constraints describing the same issue.

3.3.3 Solver

It is the solver's task to solve the constraint satisfaction problem. This happens by assigning values to each variable such that all constraints are fulfilled. Depending on the used types of constraints and parameters and other boundary conditions, different methods to solve the constraint satisfaction problem have to be chosen. Possible solving algorithms are [62]:

- algorithms for nonlinear problems:
  - backtracking
  - local consistency
  - local search
- algorithms for linear and polynomial equations and inequalities:
  - variable elimination
  - simplex algorithm to solve linear and polynomial equations and inequalities

The explicit background of these algorithms is not within the scope of this work and therefore omitted. Detailed information can be found in [62, 71].

3.4 Optimization with Simulated Annealing

Simulated annealing is a stochastic metaheuristic that mimics the crystallization of molten metal. If this crystallization happens during a slow cooling process, called annealing, a state of minimum energy is reached when the crystallization is finished. Simulated annealing was presented in 1983 by Kirkpatrick et al. [89] and adapts this natural phenomenon to optimization problems.

$$\underset{x}{\text{minimize}} \quad f(x) \tag{10}$$

The general simulated annealing algorithm for the basic optimization problem shown in Eq. 10 can be formulated as follows [90]:

1. The initial temperature T_0 and an initial value for x with objective function value $f(x)$ is set.
2. A random x^* is selected from the neighborhood of x and $f(x^*)$ is calculated.

3. The two objective function values are compared using the Metropolis criterion. If $U \leq e^{(-\frac{\Delta}{T})}$, with U being a random real number between zero and one, T being the current temperature, and $\Delta = f(x^*) - f(x)$, x is set to x^*. Otherwise, x remains unchanged.

4. Steps two and three may be repeated depending on some criterion, e.g. if x was changed to x^* or a number of iterations is reached.

5. The temperature is lowered and the process starts over from step two.

This process is repeated until a global abort criterion, for instance a total maximum number of iterations or a maximum number of iteration without change of x, is reached. The use of the Metropolis criterion in step three enables the possibility that values for x^* can be accepted even if $\Delta > 0$. This allows the algorithm to overcome local minima. Using Markov chains convergence for simulated annealing, given infinite time, was proven [90]. In order to fit the algorithm to specific tasks, the user has to define the annealing schedule, e.g. the vanilla schedule [91], and the starting temperature, the neighborhood function, and the stopping criteria.

In the context of this thesis, simulated annealing was chosen due to several reasons. First, it showed its potential in a wide range of synthesis applications, e.g. [39, 34]. Second, and more important, the variables do not have to be encoded in a fixed representation as, for instance in genetic algorithms [92]. Since different numbers of variable can be expected from alternative concepts, no generic encoding scheme for sets of variables with different sizes has to be developed.

4 Method

Using and applying the methodologies presented in Section 3, this section presents an automated approach to synthesize and simulate solution spaces for energy- and signal-based engineering tasks in an object-oriented graph-based environment [53]. As shown in Figure 8, the method comprises the following steps:

1. Definition of the metamodel
2. Topological synthesis using first-order logic and Boolean satisfiability
3. Variable assignment generation using constraint satisfaction
4. Single-objective optimization using simulated annealing

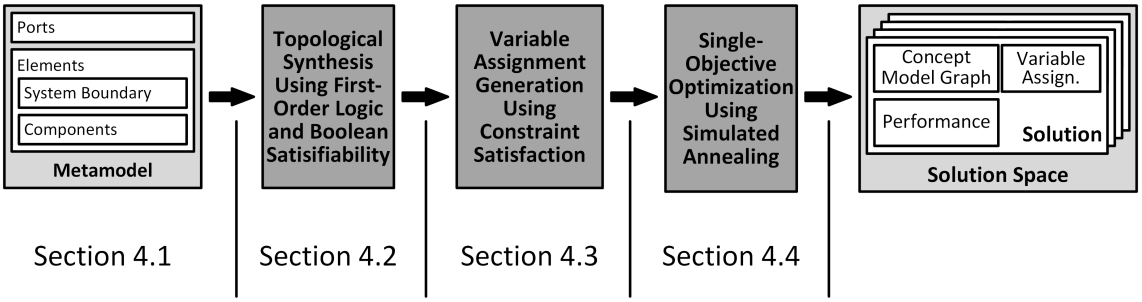


Figure 8: Structure of the method

The starting point of the method is the **metamodel**. Within the metamodel, the designer encapsulates the task specification. He or she defines the types of elements that are available for the concept generation. There are two basic different types: components, from which the concepts can be composed, and a system boundary, that represents the environment of the product to be designed. Interfaces between components and between components and the system boundary are modeled using ports. These ports represent energy and signal flows. The set of interfaces of an element is called its port configuration. Apart from the port configuration the elements have further attributes. These include design variables, constraints on those variables, constraints linking the variables with the energy and signal flows, and a partial simulation model for the components and critical operation points, a partial simulation model, and an optimization problem for the system boundary.

Representing the design task in the metamodel, i.e. the definition of the components and the system boundary, can be achieved after the main function "that represents the intended overall relationship between the inputs and the outputs of a plant, machine or assembly" [10] is defined in the beginning of the conceptual design. This overall relationship is represented in the port configuration of the system boundary which also refers to the theory of technical systems where Hubka and Eder [93] suggest understanding a technical artifact, i.e. a product, as a system that is connected to the environment by its in- and outputs.

The next step is to generate concepts by connecting the open ports of the system boundary using the defined components, such that a complete topology [94], i.e. a topology with no open ports, results. This problem is referred to as port-matching-problem (PMP). To tackle this task, the knowledge stored in the metamodel is used in the **topological synthesis using first-order logic and Boolean satisfiability**. The metamodel, and basic constraints are translated into logical expressions that describe the solution space. The basic constraints cover the following:

1. The maximum number of components to be used in the solution
2. Only ports of the same type can be connected
3. There has to be exactly one system boundary per solution
4. There must be no unconnected ports in the solution

The first constraints prohibits an infinite problem with an, therefore, infinite solving time. The second one implements the compatibility information that is founded in the port hierarchy. Constraints three and four are the basis for the actual synthesis: There has to be one system boundary that has ports which are to be connected using relation and components so that there is no more open port. The combination and connection of the components between themselves and the system boundary build the representation of the concept, a concept model graph (CMG). In order to achieve this, the logical expressions are transformed into a Boolean satisfiability problem. This is then solved by a SAT-solver, i.e. a software tool to solve Boolean satisfiability problems, and the Boolean solution (if one is found) is transformed back into a CMG. An example CMG is shown in Figure 9. By negating the discovered Boolean solution and beginning a new search for a solution, the solution space can be explored exhaustively.

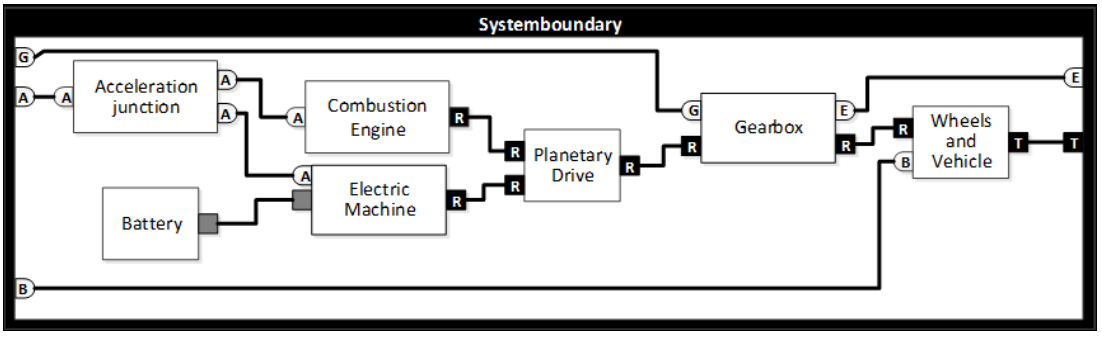


Figure 9: An example concept model graph (CMG) for a hybrid automotive powertrain

In the next step, the **variable assignment generation using constraint satisfaction**, for all generated CMGs, assignments for the design variables are generated. From the constraints that are embedded in each component and operation points that are defined in the system boundary, a constraint satisfaction problem is generated and then solved. This allows the detection of concepts that cannot fulfill the operation points. These solutions are removed from the solution space and for the rest of them, a predefined number of possible variable assignments is generated.

To find the optimal variable assignment, the last step of the method, **single-objective optimization using simulated annealing** is conducted. A simulation model is build from the partial simulation models of the components and the system boundary. This is done by retrieving the partial simulation models and connect them as the corresponding elements are connected in the CMG. The simulation model is used to evaluate the generated concept and calculate the objective function value as defined in the optimization problem in the system boundary. When the simulated annealing process is completed for every concept, the final set of solutions is finished. It consists of solutions that each contain a CMG, the optimal variable assignment and the solution's performance, i.e. the results from the simulation with the optimal variable assignment.

In the following Section 4.1, details for the metamodel, its constituent, and the metamodeling process are given. Then, Section 4.2 gives insights on the topology generation and Section 4.3 on the generation of variable assignments. Finally, Section 4.4 describes the simulated annealing process including the automated generation of simulation models.

4.1 Metamodel

The foundation for the solution space generation and evaluation is the metamodel. The ISO 11179 [95] defines a metamodel as a "data model that specifies one or more other data models". Here, it is the definition of types of available elements which are components and a system boundary to the product environment and the definition of different port types. Thus, the definition of the metamodel formalizes the task specification and the engineering problem-solving knowledge.

In the following sections, the three constructs defined in the metamodel, elements, ports, and edges are described. Apart from the constituents and their meaning, also guidelines for the metamodeling are given.

4.1.1 Ports

Ports, i.e. instances of the port types defined in the metamodel, are interfaces from one element to another. There are in total two basic types of ports: flow-ports and conceptual ports. Flow ports, as the name suggests, model a flow between two elements. Two types of flows are divided: energy and signal flow. Conceptual ports model more abstract relations, e.g. dependencies.

Ports, in general, have two attributes. One is an identifier that is used to identify a specific port within a port configuration and constraints of an element (see 4.1.2). The other defines the port's direction. It can have two different values: *in* or *out*. For energy flow ports, the value defines the reference direction of the energy flow, similar to the half-arrow in bond graphs [87], for signal flow and conceptual ports, the obligatory connection direction is set here.

Table 1: Power and energy variables in various energy domains (reproduced from [87])

Energy domain	Effort e	Flow f
Translational mechanics	Force F [N]	Velocity v [m/s]
Rotational mechanics	Angular momentum M [Nm]	Angular velocity ω [rad/s]
Electrical	Voltage u [V]	Current i [A]
Hydraulic	Total pressure p [N/m^2]	Volume flow rate Q [m^3/s]
Thermodynamic	Temperature T [K]	Entropy flow rate $\dot{S}$ [J/K/s]
Chemical	Chemical potential μ [J/mol]	Molar flow $\dot{N}$ [mol/s]

Additional to these basic attributes that are inherited from the basic port type, energy flow ports have further attributes to specify the magnitude of the associated energy flow. Following the bond graph theory, energy flow $\dot{E}(t)$, i.e. power $P(t)$, can be divided into effort $e(t)$ and flow $f(t)$, see Eq. 11. Tab 1 shows flow and effort definitions for different

energy domains.

$$\dot{E}(t) = P(t) = e(t) \cdot f(t) \tag{11}$$

For each energy flow port, the magnitude of each, flow and effort, have to be constrained. As shown in Eq. 12, both values have to remain between two borders. These two constraints are called flow-range and effort-range.

$$\begin{aligned} e_{min} \leq e(t) \leq e_{max} \\ f_{min} \leq f(t) \leq f_{max} \end{aligned} \tag{12}$$

The taxonomy of inheritance in which the ports are defined is shown in Fig. 10. The different attributes are depicted in curly brackets. Three vertical dots indicate where additional port types, e.g. further energetic domains, specific signals, dependencies etc., can be defined as needed. Due to the inheritance in the port hierarchy, the attributes are only defined once and then inherited.

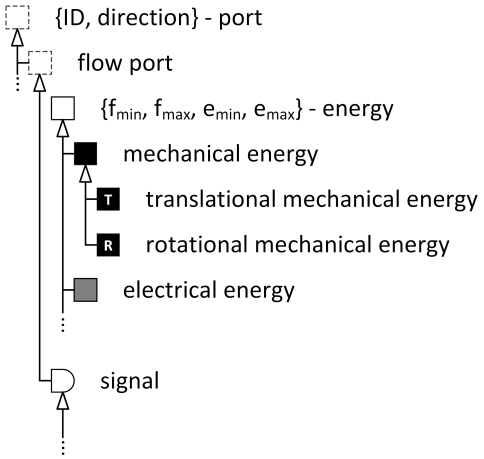


Figure 10: Port hierarchy showing different types of flow values.

Material flow can only be modeled if it can be represented as chemical energy flow as shown in Table 1. This is due to the use of bond graph-based simulation that is based on the exchange of energy. However, material flow can only be modeled conceptually. An example for this will be shown in the case study on chemical process engineering in Section 6.1.

4.1.2 Elements

In the metamodel two basic types of elements are defined: a) the system boundary of the product that is to be designed and b) components, i.e. physical structures and parts. An exemplary illustration of both these element types is shown in Fig. 11.

Every element has attributes that specify certain characteristics and properties. The attributes are grouped in different categories. There are *general attributes*, *design variables and constraints*, and *mapping to partial simulation models* which relate to both types of elements, and an *optimization model*, that is defined as attribute of the system boundary.

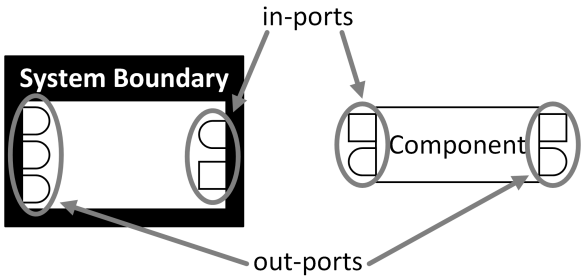


Figure 11: An exemplary illustration of components and system boundaries with the positioning of their in- and out-ports. This exact positioning is applied all throughout this thesis.

The different categories are described in the following paragraphs.

General Attributes The general attributes are a unique ID that allows the identification of every instance and a port configuration. This configuration determines which ports the element has. An example gearbox component with its port configuration are shown in Fig. 12. It has one signal in-port that defines which gear to engage (G), one signal out-port transmitting the rotational speed at the input shaft (E) and two mechanical rotational energy ports (R): one in- and one out-port representing the input and output shaft of the gearbox. As shown in Figure 11 and 12, throughout this thesis, the in-ports are always located on the left and the out-ports always on the right side of the element.

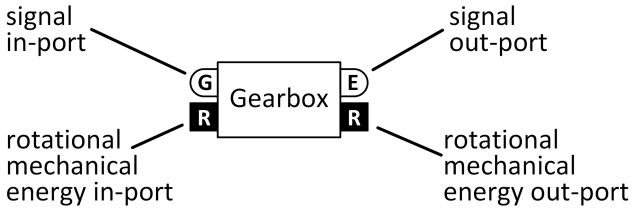


Figure 12: The port configuration for a gearbox component.

Design Variables and Constraints As mentioned in the beginning of Section 4, assignments for design variables are set by generating and solving constraint satisfaction problems. To enable this, constants, variables, input-output-relations, and additional constraints can be defined for components.

- *Constants* are constant values that are specific for the particular component.
- *Variables* are numeric design variables that are to be defined for the final design. For each variable, a domain has to be defined. This domain consists of the minimum and maximum value between which the assigned value has to be chosen.
- *Input-output-relations* define the equations or inequalities that connect the input and output energy- and signal-flows of the component. These energy flows are divided in flow and effort, as introduced in Section 3.2. Every input-output-relation has to

Constants	**Variables**
$\eta = 0.98$	$0.6 \leq i_m \leq 5.0, m \in \{1, 2, \ldots, 6\}$
Additional constraints	**Input-output-relations**
$i_1 > i_2$ $i_2 > i_3$ $\vdots$	$\omega_I \frac{1}{i_G} = \omega_O$ $T_I \, i_G \, \eta = T_O$ $E = \omega_I$

Figure 13: The gearbox component's constants, variables, additional constraints, and input-output-relations. **G**,**E**,**I**,**O** are the identifiers of the associated ports.

include at least one variable that is related to an in- or out-port. Whereas simulation models are time-dependent, input-output-relations are static and, therefore, time-independent.

- *Additional constraints* define dependencies between the component's design variables and constants. In contrast to the input-output-relations, effort and flow or signal variables must not be used in these constraints.

For the gearbox component, the four different parts are shown in Fig. 13. Here, the only energy flow ports used are rotational mechanical. As shown in Tab. 1, rotational mechanical power is distributed in rotational speed ω_n and torque T_n with n being the identifier of the associated port (*G*,*E*,*I* or *O*). When a signal port is used, the port's identifier is used directly (*E*).

In the system boundary, no constants, variables, constraints, and relations are defined, but so-called operation points. Operation points are typical and/or critical operational states that the design has to fulfill. For each operation point, the flow and effort values for every energy flow in-port and out-port and the signals from and to the system boundary can be defined.

Mapping to Partial Simulation Models In order to allow a performance evaluation of the generated concepts, every element has to be mapped to a bond graph-based partial simulation model. Such partial models are not closed and, therefore, contain junctions where they can be connected to other partial simulation models, so-called open connectors.

To build partial simulation models for the components and the system boundary, the following guidelines have to be followed:

1. Open connectors of the partial simulation model have to fit the element's port configuration

2. A resistor-capacitor-connection has to be modeled in front of connectors in the partial simulation model that map to out-ports in the port configuration
3. After connectors that map to in-ports are modeled an inertia has to follow
4. Modelwide definitions have to be defined in the partial simulation model that is linked to the element representing the system boundary

The first guideline relates to the fact that in there are to be no open connectors in a well-defined simulation model. Since, as mentioned before, one of the constraints for the topological synthesis (more in Section 4.2) is that in a CMG no unconnected ports are allowed, the one-to-one mapping between the ports in the port configuration and the open connectors in the simulation model assures a well defined simulation model.

The second and third guideline are needed to assure a valid causality in the simulation model that is generated from a CMG.

As mentioned in Section 3.2, in order to get a valid causality, every bond in a bond graph must define flow at one and effort at the other end of the bond and conform to the obligatory causalities that certain elements dictate. That means, that the bond graph constituents cannot be connected arbitrarily. An example for an invalid connection in a bond graph is shown at the top of figure 14: the series connection of two inertias. The causality of the left inertia is correct and forwarded over the 1-junction to the right inertia where a wrong causality results (the causality stroke has always to be on the side of the inertia).

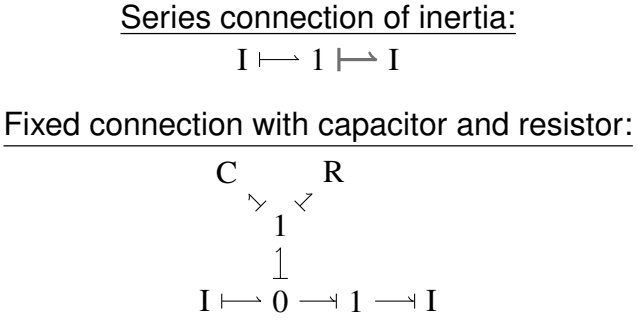


Figure 14: Compatibility issues between two elements of one kind and their solution. The erroneous bond in the upper example is highlighted in thick gray.

Considering the example of the series connection of the two inertias in the mechanical translational domain, two masses would be connected by a push rod that could be represented by a CMG where the masses are modeled by two components that are connected via an edge between mechanical translational ports. A real world push rod has a spring coefficient, a damping coefficient, and an inertia itself. Cutting between the spring (capacitor) damper (resistor) connection and the inertia in the corresponding bond graph, distributes the model of the push-rod in two different partial simulation models. In the bottom part of Figure 14, such a capacitor-resistor-connection is included and a valid causality is established. This is transferred to every energy domain and is applied at each port. So, the connection of two compatible ports results in a valid connection between the two related partial simulation models which, in the end, leads to a simulation model with

a well defined causality.

According to the fourth guideline, global variables and definitions, for instance fluid properties or environmental data as, for instance, the ambient temperature, have to be defined in the system boundary. As mentioned before, every CMG has exactly one instance of this element type. Thus, duplicate global definitions are being prevented.

In Figure 15, a gearbox element with its associated partial simulation model is shown. Using their unique identifiers, each port is linked to one open connector in the partial simulation model. In Figure 15 this mapping between connectors and ports is illustrated using dashed gray lines. The shaft-element highlighted on the right side of the partial simulation model represents the capacitor and resistor that was added following the presented guidelines. On the very left of the partial simulation model in Figure 15, the temperature of the oil in the gearbox is fixed to the parameter k. This is a) done to simplify the model (here, the warming and waste heat were not considered) and b) necessary so that the open connectors of the partial simulation model meet the port configuration of the graph element in the metamodel.

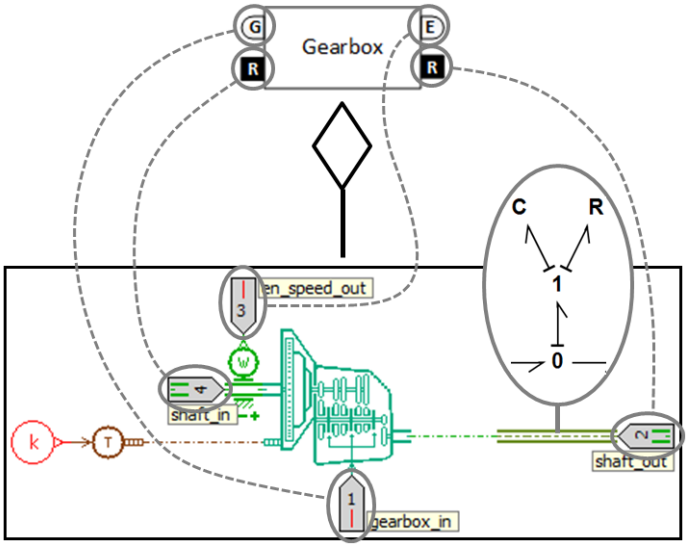


Figure 15: Graph symbol and partial simulation model. The dashed gray lines illustrate the mapping between the connectors in the simulation model and the ports in the graph model.

Figure 16 illustrates a system boundary and its partial simulation model. This element represents the system boundary for an automotive powertrain as it is used in Section 6.3. A brake- (Ⓑ), acceleration- (Ⓐ), and gear-signal (Ⓖ) are provided and to be turned into translational mechanical energy (T) and an engine speed signal (Ⓔ). As defined in the guidelines, on the upper left of the partial simulation model, the model-wide definitions are set (mission profile, ambient data, and fluid properties).

Optimization Model In each system boundary an optimization model has to be defined as shown in Eq. 13. $\mathbf{x}$ refers to a variable assignment for the concepts design variables that has to be taken out of V, which is the set of variable assignments that is created using constraint satisfaction (see Section 4.3 for details). The size of V has to

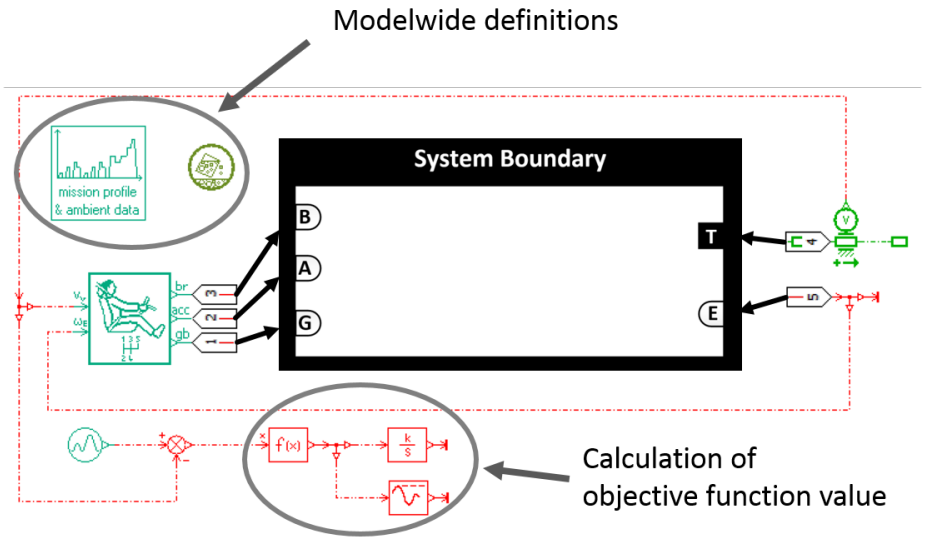


Figure 16: An example system boundary for hybrid powertrains. Around the graph symbol the partial simulation model is illustrated.

be defined to y which controls how many variable assignments are to be generated, as well as the amplifier k that is needed to be set to z for the optimization (further details in Section 4.4). $f(\mathbf{x})$ is the objective function. It is calculated during the simulation in the partial simulation model of the system boundary. In the example shown in Figure 16, it is the deviation of a given speed profile that is derived in the lower part of the model.

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in V, \\ & \mathit{size}(V) = y, \\ & k = z \end{aligned} \tag{13}$$

4.1.3 Edges

To connecting two ports of different elements an undirected *relation*-edge-type is defined. An edge of this type can only be set if two preconditions are met: a) both ports are of the same type and b) the value of the direction attribute of one of the ports is *in* and the other's is *out*. Further, if the two energy flow ports are to be connected, the flow and effort ranges of both ports must each intersect. An exemplary use of a *relation*-edge is shown in Figure 17, where it is used to connect an electric machine and a gearbox.

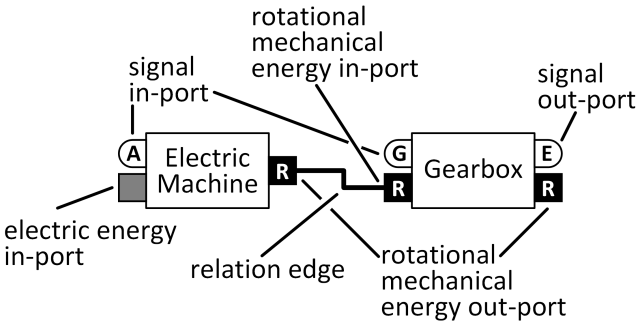


Figure 17: A connection between an electric machine and a gearbox as an example for elements and their connections. Throughout this thesis, the in-ports are always located on the left-hand side and the out-ports always on the right-hand side of the element.

4.2 Topological Synthesis Using First-Order Logic and Boolean Satisfiability

Figure 18 presents a process overview for synthesizing concept topologies using first-order logic and Boolean satisfiability. The process builds on a metamodel defined after Section 4.1.

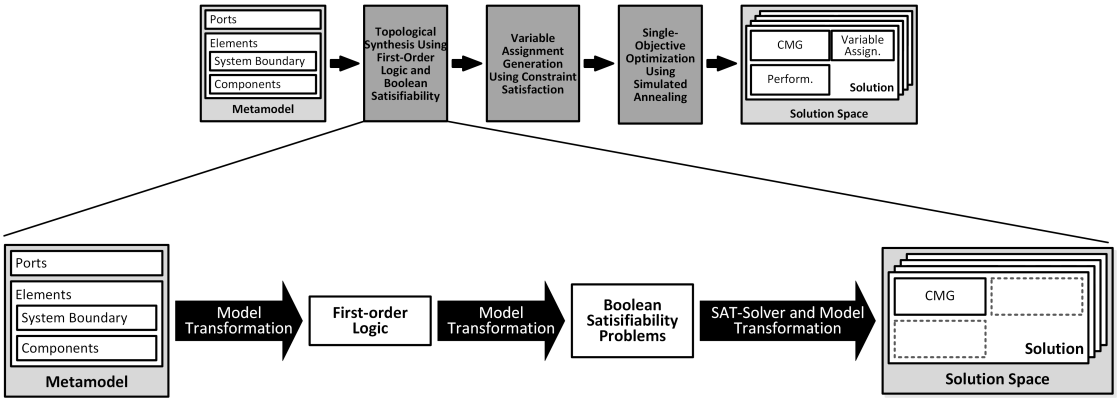


Figure 18: Topological synthesis process overview.

Following the process shown in Figure 18, a first-order logic description is generated from the metamodel and the general constraints described on page 30. All needed information is extracted and translated into logical expressions. Once the logic is completed, it is transformed into a Boolean satisfiability problem (SAT). A SAT-solver checks if the problem is solvable, given the maximum number of components and, if so, returns a solution. This Boolean solution is interpreted and a model to the first order logic is generated. This model is transformed into a solution containing a CMG within the initial object-oriented graph-based environment. The main parts of the process are described in the following sections.

4.2.1 First-order Logic Representation of the Metamodel

In this subsection, the metamodel is turned into a first-order logic description. This first-order logic describes everything regarding solving the PMP that is defined by system boundary, ports, and components. A new general taxonomy is defined. As shown in Figure 19, a *thing* class is introduced. This *thing* class is a fixed part of ontologies [96]. It is the superclass of all classes that may be used in the solution graph. As descendants of this class, two new classes are introduced: *element* and *port*. Further, the *port*-class has two descendants: *port_in* and *port_out*.

The element types and port types from the metamodel are integrated into the general taxonomy (Figure 19). The original hierarchy of inheritance that structures the elements in the original metamodel is not transferred into this taxonomy. Instead, the components are organized in categories that are created based on the total number of in- and out-ports of each component.

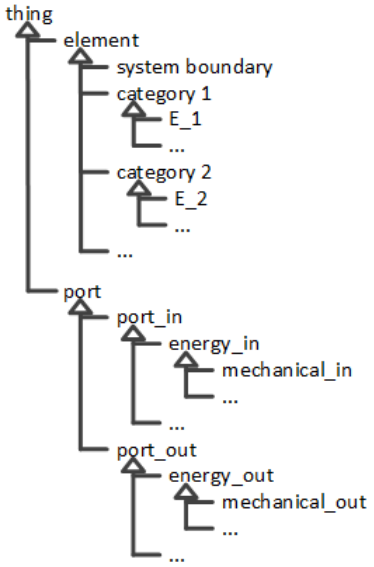


Figure 19: First-order logic general taxonomy with example ports and elements

An example taxonomy is shown in Figure 19. Since in the logic, ports and elements do not have attributes, the value of the direction attribute that determines the direction of the port as described in Section 4.1.1 has to be reflected in the port name. This is done by adding a suffix (*_in* or *_out*) to the original port name. Apart from the components being organized in categories, the system boundary element is directly included as element. This element embodies the PMP and its port configuration.

One-digit predicates are derived from each class. Those predicates return *true* if its argument is an instance of the corresponding class. One example from the hierarchy in Figure 19 is the predicate $E_1(x)$. Next, the taxonomy's reduced inheritance has to be defined using facts. The first fact (Eq. 14) assures that all elements used within the solution graph are defined in the metamodel. The following facts implement the inheritance (for example Eq. 15). They are all similar, stating that every building block x that is of a specific type is also an instance of its super type.

$$\forall x(\textit{thing}(x)) \tag{14}$$

$$\forall x(\textit{element}(x) \rightarrow \textit{thing}(x)) \tag{15}$$

The next step is to transfer each element's port configuration into a logical description. To do this, two predicates are needed: *hasport(x,y)* and *identical(x,y)* (cf. Eq. 16 and 17). As shown in Figure 20, *hasport(x,y)* is *true* if a port y is part of the port configuration of an element x. Figure 20, also illustrates *identical(x,y)*. This predicate returns *true* if both variables, x and y, both point to the same item. Since the model is a CMG, an item is either an element or a port. Variables cannot point to edges due to them being modeled as predicate, i.e. *connected(x,y)* which is described later, and not things.

$$\textit{hasport}(x, y) \tag{16}$$
$$\textit{identical}(x, y) \tag{17}$$
$$\tag{18}$$

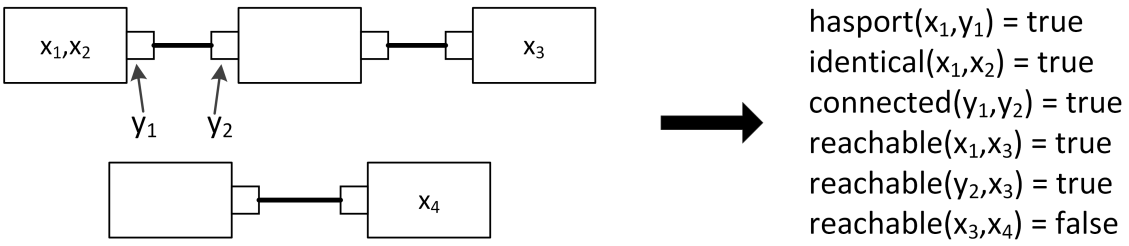


Figure 20: The evaluation of the different predicates on a generic graph example.

The facts describing an element's port configuration are built as follows: First, all things are restricted to the ones meant to be treated in the fact. For the example element E_1 (Figure 21) this is $\forall x\ E_1(x)$. If this is true the following statement is implicated: $\exists y_1\ P_{1,in}(y_1) \wedge \exists y_2\ P_{1,out}(y_2)$. Also, y_1 and y_2 have to be linked to the element E_1: $\textit{hasport}(x, y_1) \wedge \textit{hasport}(x, y_2)$. As it is still allowed to add any other port, since $\exists$ only implies "one or more" and there are no statements considering the other port types, another statement has to be added: $\neg\exists z\ \textit{port}(z) \wedge \neg\textit{identical}(z, y_1) \wedge \neg\textit{identical}(z, y_2) \wedge \textit{hasport}(x, z)$. This statement means that there is no port z that is not identical to either y_1 or y_2 and also belongs to the element x. Using this structure, also port configurations with more ports of the same type can be realized by assuring that they are not identical. The resulting fact for E_1 is shown in Eq. 19. Removing the need of two ports of the same type to be not identical also allows the dynamic definitions of ports, for instance if an element has one or more ports of one type, so-called port ranges.

$$\begin{aligned}\forall x(E_1(x) \rightarrow (\exists y_1(P_{1,in}(y_1) \wedge \exists y_2(P_{1,out}(y_2)\\ \wedge \neg\exists z(\textit{port}(z) \wedge \neg\textit{identical}(z, y_1) \wedge \neg\textit{identical}(z, y_2) \wedge \textit{hasport}(x, z))))))\end{aligned} \tag{19}$$

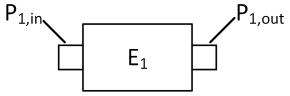


Figure 21: Example element E_1

Finally, it has to be assured that no port belongs to two different elements or is isolated. So, the fact in Eq. 20 is defined. It says that for every port there is only one element that it belongs to.

$$\begin{aligned}\forall y(\textit{port}(y) \rightarrow (\exists x_1(\textit{element}(x_1) \wedge \neg\exists x_2(\textit{element}(x_2)\\ \wedge \neg\textit{identical}(x_1, x_2) \wedge \textit{hasport}(x_1, y) \wedge \textit{hasport}(x_2, y)))))\end{aligned} \tag{20}$$

All information from the metamodel is integrated into the logical description. Next, the following constraints are integrated to produce valid CMGs:

1. There are no unconnected ports.
2. Exactly one system boundary element is part of the solution CMG.

The first constraint corresponds to the need for a complete topology [94]. Such topologies do not have any loose ends, i.e. open ports. For ports that model an energy flow, this also corresponds to the conservation of energy. As shown in the example concept model graph in Figure 9 on page 30, a concept topology "fills" the system boundary and connects the ports to the product environment using the components defined in the metamodel. Therefore, exactly one system boundary is needed to be the frame for each generated concept. This is assured by constraint two.

$$\mathit{connected}(x, y) \tag{21}$$

To model these constraints, further predicates are needed. The first is called *connected(x,y)* (cf. Eq. 21). As shown in Figure 20, this predicate returns *true* if x and y are ports that are connected by an undirected edge. As mentioned in the first constraint, all ports have to be connected. So, Eq. 22 results.

$$\forall y_1(\mathit{port}(y_1) \rightarrow (\exists y_2(\mathit{port}(y_1) \wedge \mathit{connected}(y_1, y_2)))) \tag{22}$$

$$\mathit{types\_match}(x, y) \tag{23}$$

$$\mathit{directions\_match}(x, y) \tag{24}$$

However, there are more constraints that have to be assured for two connected building blocks since not all building blocks in the solution are allowed to be connected. If *connected(x,y)* is true, x and y both have to be ports, they are not allowed to be identical and both their types and directions have to match. In order to achieve this, two further predicates are introduced: *types_match(x,y)* (cf. Eq. 23) and *directions_match(x,y)* (cf. Eq. 24). The predicate *types_match(x,y)* returns true if the types of both ports x and y are compatible, so if y is of the same type as x. *directions_match(x,y)* returns true when the directions of both ports x and y are matching. That means that neither two *port_in*-ports or two *port_out*-ports are connected. All other combinations are valid. This is realized in Eq. 25.

$$\begin{aligned}\forall y_1 \forall y_2(\mathit{connected}(y_1, y_2) \rightarrow (&\mathit{port}(y_1) \wedge \mathit{port}(y_2) \\ &\wedge \neg \mathit{identical}(y_1, y_2) \wedge \mathit{types\_match}(y_1, y_2) \\ &\wedge \mathit{directions\_match}(y_1, y_2))\end{aligned} \tag{25}$$

The second constraint is also transferred into a logical statement. The system boundary element has to be part of the graph since if there are no unconnected ports it then contains

a topology that fits to the original PMP. Eq. 27 shows the fact assuring the system boundary element's occurrence and states that one element in the solution is an instance of the problem class. The second conjunction of the statement assures that there is exactly one system boundary element. Additionally, it has to be guaranteed that the system boundary element must be reachable from any other element in the graph. That means that no isolated subgraph islands are allowed to appear in the solution graph. As shown in Figure 20 the predicate *reachable*(x, y) (cf. Eq. 26) embodies this correlation. *reachable*(x, y) is *true*, if the corresponding things of x and y are directly or indirectly connected by relations. In this method, every thing must be able to reach the system boundary element. So, the third conjunction in Eq. 27 results.

$$\textit{reachable}(x, y) \tag{26}$$

$$\exists x_1(\textit{system\_bountary}(x_1) \wedge \neg\exists x_2(\textit{system\_bountary}(x_2) \wedge \neg\textit{identical}(x_1, x_2)) \wedge \forall x_3(\neg\textit{identical}(x_3, x_1) \rightarrow \textit{reachable}(x_3, x_1))) \tag{27}$$

Now, the complete first-order logic for the reduced metamodel can be generated based on the equations 14 to 27. The next step is to transform this logic into a Boolean satisfiability (SAT) problem.

4.2.2 Conversion to SAT and Solving

As mentioned in Section 3.1.3, to translate first-order logic descriptions to a finite SAT problem, a scope has to be defined. The scope is defined as the number of elements from each category and the total number of ports in the solution graph. The possible scopes are determined from the metamodel. As described in the last section, the components are sorted into categories. These categories differ in the number of in- and out-ports. The sorting is realized by abstracting each element's port configuration. For instance, the example element shown in Figure 21 is part of the category for elements with one in- and one out-port.

The open port configuration of the system boundary element can only be closed by specific sets of instances of the abstract categories. In order to find all possible scopes, all valid assignment for the variables $n_i \in \mathbb{Z}^*$ with $i \in \{1, 2, \ldots, c\}$ and c being the number of categories are generated using breadth-first search. Valid assignments have to satisfy Eqs. 28 and 29 where $z_{P_{in},i}$ is the number of in-ports and $z_{P_{out},i}$ is the number of out-ports for category i and $z_{P_{in},sb}$ and $z_{P_{out},sb}$ are the number of in- and out-ports for the system boundary.

$$\sum_{i=1}^{c} (n_i \cdot (z_{P_{in},i} - z_{P_{out},i})) + z_{P_{in},sb} - z_{P_{out},sb} = 0 \tag{28}$$

$$\sum_{i=1}^{c} n_i \leq n_{max} \tag{29}$$

For each assignment of n_i, the total number of ports z_P is calculated using Eq. 30.

$$z_P = \sum_{i=1}^{c} \left(n_i \cdot \left(z_{P_{in},i} + z_{P_{out},i}\right)\right) + z_{P_{in},sb} + z_{P_{out},sb} \tag{30}$$

Each calculated scope is then solved as described in the following paragraphs.

To solve the problem for the defined scopes, the process introduced by Jackson [86] and Torlak [85] as described in Section 3.1 is used to transform the scope and the first-order logic description of the metamodel into a Boolean satisfiability problem. The resulting Boolean formula that is then solved by a SAT-solver. If the solver finds a Boolean solution, it is translated into a model to the first-order logic description. This model is then further translated to a CMG in the original object-oriented graph based framework. Therefore, the element types are instantiated that correspond to those in the first-order model. The ports are also connected as defined in the model. The result is a solution containing a CMG for the initial design task.

4.2.3 Energy and Signal Conservation check

The final step of the topological synthesis is to check the signal and energy conservation within the model. Therefore, a total of four checks are performed:

1. Is an energy source (element without energy inports and at least one energy outport) or the system boundary reachable from every energy inport?
2. Is an energy sink (element without energy outports and at least one energy inport) or the system boundary reachable from every energy outport?
3. Is a signal source (element without signal inports and at least one signal outport) or the system boundary reachable from every signal inport?
4. Is a signal sink (element without signal outports and at least one signal inport) or the system boundary reachable from every signal outport?

If all four checks are true, energy and signal consistency is given. Thereby, so-called closed circuits in the simulation model are avoided. Solutions that do not fulfill one or more of the four checks are removed from the solution space.

Instead of checking these conditions after the generation it would also be possible to include further facts in the first-order logic description by defining new predicates and constraints. However, this method for topological synthesis is designed to work with general PMPs also outside the domain of mechanical- and signal-based systems (an example is presented in the case study on chemical process engineering in Section 6.1). Extending the logical description limits the range of applications, since the existence of energy and signal ports (at least in the definition) would have to be presumed.

For the solutions that remain after the energy and signal conservation check the next step is the generation of variable assignments for the design variables. The details for this part of the method is given in the following Section.

4.3 Variable Assignments Generation Using Constraint Satisfaction

The target of this step is to generate variable assignments for the design variables that are defined in the components of the metamodel. For each solution, the associated CMG is analyzed and a constraint satisfaction problem (CSP) is created and then solved to generate the predefined number of variable assignments that are to be available for the simulated annealing process that is described in Section 4.4. In Section 4.3.1 the automated generation of a CSP based on a CMG is described before Section 4.3.2 discusses solving it.

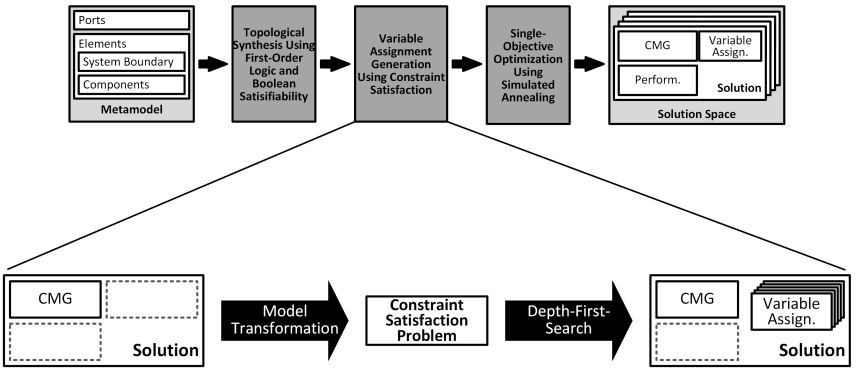


Figure 22: Process overview for the generation of alternative variable assignments

4.3.1 Constraint Satisfaction Problem Generation

From the given CMG in the solution, certain attributes are gathered from the used components and the system boundary. The variables that need assignment and their maximum and minimum values and the constants of every component are identified. Also, the additional constraints from each component are collected. Following this, the operation points are pulled from the system boundary. For each operation point, the following steps are carried out:

1. A variable is defined for every signal port representing the signal that is transmitted.
2. For every energy port one flow- and one effort-variable is defined within the particular flow and effort range, respectively.
3. The input-output relations from each component in the CMG are linked to these signal-, flow-, and/or effort-variables.
4. From the relations in the CMG, the connected signal-, flow, and effort-variables are set to be equal.
5. The values that are fixed for the ports of the system boundary are fixed to the associated port-variables.

Given the gearbox component and its variables and constraints (cf. Figure 13) as sole component in a system boundary as shown in Figure 23. The system boundary defines

two operation points O_1 and O_2 each setting the gear signal out-port (**S**) value to S_{O_1} and S_{O_2} and the flow- and effort-values for the rotational mechanical in-port (**Y**) to ω_{Y,O_1}, M_{Y,O_1} and ω_{Y,O_2}, M_{Y,O_2}, respectively.

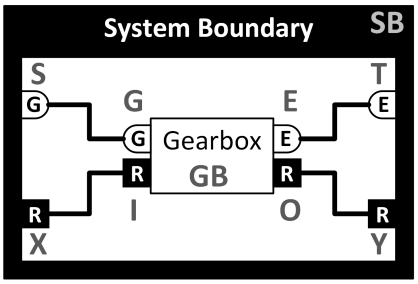


Figure 23: An example CMG with a system boundary **SB** and a gearbox **GB**. **S**,**T**,**X**,**Y** being the identifier of the system boundary's ports and **G**,**E**,**I**,**O** being the identifier of the gearbox' ports

Following the given procedure, the constant (Eq. 31) and design variables (Eq. 32) from the gearbox are included in the CSP. As are the additional constraints (Eq. 33). To link the variables and constants to the corresponding element, here the gearbox component, they are indexed with the element's identifier, here GB:

$$\eta_{\mathbf{GB}} = 0.98 \tag{31}$$

$$0.6 \leq i_{m,\mathbf{GB}} \leq 5.0, m \in \{1, 2, \ldots, 6\} \tag{32}$$

$$i_{1,\mathbf{GB}} > i_{2,\mathbf{GB}}; i_{2,\mathbf{GB}} > i_{3,\mathbf{GB}}; \ldots \tag{33}$$

Steps one through five are carried out for each operation point. Eq. 34 to 55 show the resulting definitions and constraints for operation point 1. $\omega_{PID,\mathbf{EID},OP_i}$ represents the flow variable of rotational mechanical power and $M_{PID,\mathbf{EID},OP_i}$ the effort variable. Here, *PID* is the identifier of the associated port, **EID** is the identifier of the associated element, and OP_i identifies operation point number i. $\omega_{PID,\mathbf{EID},min}$, $\omega_{PID,\mathbf{EID},max}$, $M_{PID,\mathbf{EID},min}$, and $M_{PID,\mathbf{EID},max}$ are the maximum and minimum values for the flow- and effort-variables for port *PID* at element **EID** as defined the port's effort- and flow-ranges (cf. Eq. 12 on page 33). As mentioned before, the signals are directly linked to the signal ports' identifier. Therefore, $PID_{\mathbf{EID},OP_i}$ represents the signal flow at the port with identifier *PID* associated to the element with the identifier **EID** at operation point number i. Finally, elements' variables and constants are linked to the associated elements using the element's identifier. Here, this is the case for the variable $i_{\mathbf{GB}}$ and the constant $\eta_{\mathbf{GB}}$.

1.

$$S_{\mathbf{SB},OP_1}, G_{\mathbf{GB},OP_1} \in \mathbb{N} \tag{34}$$

$$E_{\mathbf{SB},OP_1}, T_{\mathbf{GB},OP_1} \in \mathbb{R} \tag{35}$$

2.

$$\omega_{X,\mathbf{SB},min} \leq \omega_{X,\mathbf{SB},OP_1} \leq \omega_{X,\mathbf{SB},max} \tag{36}$$
$$\omega_{Y,\mathbf{SB},min} \leq \omega_{Y,\mathbf{SB},OP_1} \leq \omega_{Y,\mathbf{SB},max} \tag{37}$$
$$\omega_{I,\mathbf{GB},min} \leq \omega_{I,\mathbf{GB},OP_1} \leq \omega_{I,\mathbf{GB},max} \tag{38}$$
$$\omega_{O,\mathbf{GB},min} \leq \omega_{O,\mathbf{GB},OP_1} \leq \omega_{O,\mathbf{GB},max} \tag{39}$$
$$M_{X,\mathbf{SB},min} \leq M_{X,\mathbf{SB},OP_1} \leq M_{X,\mathbf{SB},max} \tag{40}$$
$$M_{Y,\mathbf{SB},min} \leq M_{Y,\mathbf{SB},OP_1} \leq M_{Y,\mathbf{SB},max} \tag{41}$$
$$M_{I,\mathbf{GB},min} \leq M_{I,\mathbf{GB},OP_1} \leq M_{I,\mathbf{GB},max} \tag{42}$$
$$M_{O,\mathbf{GB},min} \leq M_{O,\mathbf{GB},OP_1} \leq M_{O,\mathbf{GB},max} \tag{43}$$

3.

$$\omega_{I,\mathbf{GB},OP_1} \cdot \frac{1}{i_{G_{\mathbf{GB},OP_1},\mathbf{GB}}} = \omega_{O,\mathbf{GB},OP_1} \tag{44}$$
$$M_{I,\mathbf{GB},OP_1} \cdot i_{G_{\mathbf{GB},OP_1},\mathbf{GB}} \cdot \eta_{\mathbf{GB}} = M_{O,\mathbf{GB},OP_1} \tag{45}$$
$$E_{\mathbf{SB},OP_1} = \omega_{I,\mathbf{GB},OP_1} \tag{46}$$

4.

$$S_{\mathbf{SB},OP_1} = G_{\mathbf{GB},OP_1} \tag{47}$$
$$E_{\mathbf{SB},OP_1} = T_{\mathbf{GB},OP_1} \tag{48}$$
$$\omega_{X,\mathbf{SB},OP_1} = \omega_{I,\mathbf{GB},OP_1} \tag{49}$$
$$\omega_{O,\mathbf{GB},OP_1} = \omega_{Y,\mathbf{SB},OP_1} \tag{50}$$
$$M_{X,\mathbf{SB},OP_1} = M_{I,\mathbf{GB},OP_1} \tag{51}$$
$$M_{O,\mathbf{GB},OP_1} = M_{Y,\mathbf{SB},OP_1} \tag{52}$$

5.

$$S_{\mathbf{SB},OP_1} = S_{O_1} \tag{53}$$
$$\omega_{Y,\mathbf{SB},OP_1} = \omega_{Y,O_1} \tag{54}$$
$$M_{Y,\mathbf{SB},OP_1} = M_{Y,O_1} \tag{55}$$

The same number of definitions and constraints has to be defined for operation point O_2. Then, the CSP for this example is completely defined.

4.3.2 Constraint Satisfaction Problem Solving

The CSP is then solved using depth first search as presented in Section 3.3.3. After a basic constraint propagation linking the directly connected variables, the different variables are set consecutively to random values in their ranges which are updated as soon as a new variable is defined. This stochastic depth first search is carried out until the predefined number of variable assignments is generated. If no variable assignments can be found, the solution is removed from the solution space. This set of assignments in combination with the CMG is the base for the simulated annealing process which is described in the next section.

4.4 Single-Objective Optimization Using Simulated Annealing

The last step of the method, intends to find the optimal variable assignment regarding the minimization of the objective function defined in the system boundary. This enables a better comparison of the generated concepts and their performances, since after the optimization, every concept can be assessed. As shown in Figure 24, the input of this part of the approach is a solution that contains a CMG and a set of variable assignments that were generated as described in Section 4.3. The simulated annealing process starting with a random variable assignment searches for a solution with the minimum objective function value. To evaluate the performance of the concept, a simulation model is built based on the CMG and parameterized according to the variable assignment that is to be investigated. The simulation is performed and the objective function value is retrieved from the simulation results. The detailed process is shown in Figure 25 and described in more detail in the following paragraph.

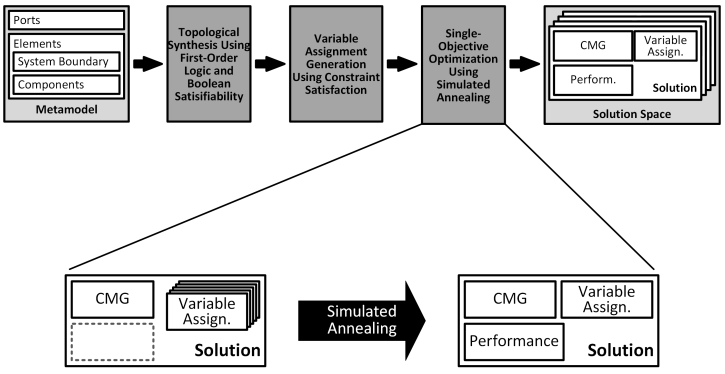


Figure 24: Input and output of the simulated annealing process.

As shown in Figure 25, the process starts with the definition of basic parameters. The initial temperature T_0 is set to 100 and the final temperature T_n is set to 0.1. During the simulated annealing process, the temperature is decreased using a vanilla schedule [91] that is further described in the course of this section. Two counters, i for the outer loop and i_{small} for the inner loop, are initialized and set to 0. The maximum number of iterations of the outer loop n, the maximum number of iterations of the inner loop m, and the amplifier k are defined by the user via the system boundary in the metamodel. The amplifier k will be described more detailed in the course of this section.

A random variable assignment *VA* is selected from the set of variable assignments in the solution. In order to evaluate it, a simulation model is built based on the CMG. Figure 26 schematically illustrates the generation of the simulation model, here using the bond graph-based commercial simulation software LMS Imagine.Lab Amesim (cf. Section 5.2.4). The partial simulation models are retrieved from the components and system boundary in the CMG and connected according to the connections in the CMG. The design variables defined in the variable assignment *VA*, in the example in Figure 26 the maximum power of the electric maching P_{max}, the gear ratios in the gearbox $i_1, i_2, \ldots i_6$ and the gear ratios of the differential in the wheels and vehicle component i_{diff} are transferred to the simulation model, before the model is compiled and the simulation is executed. The

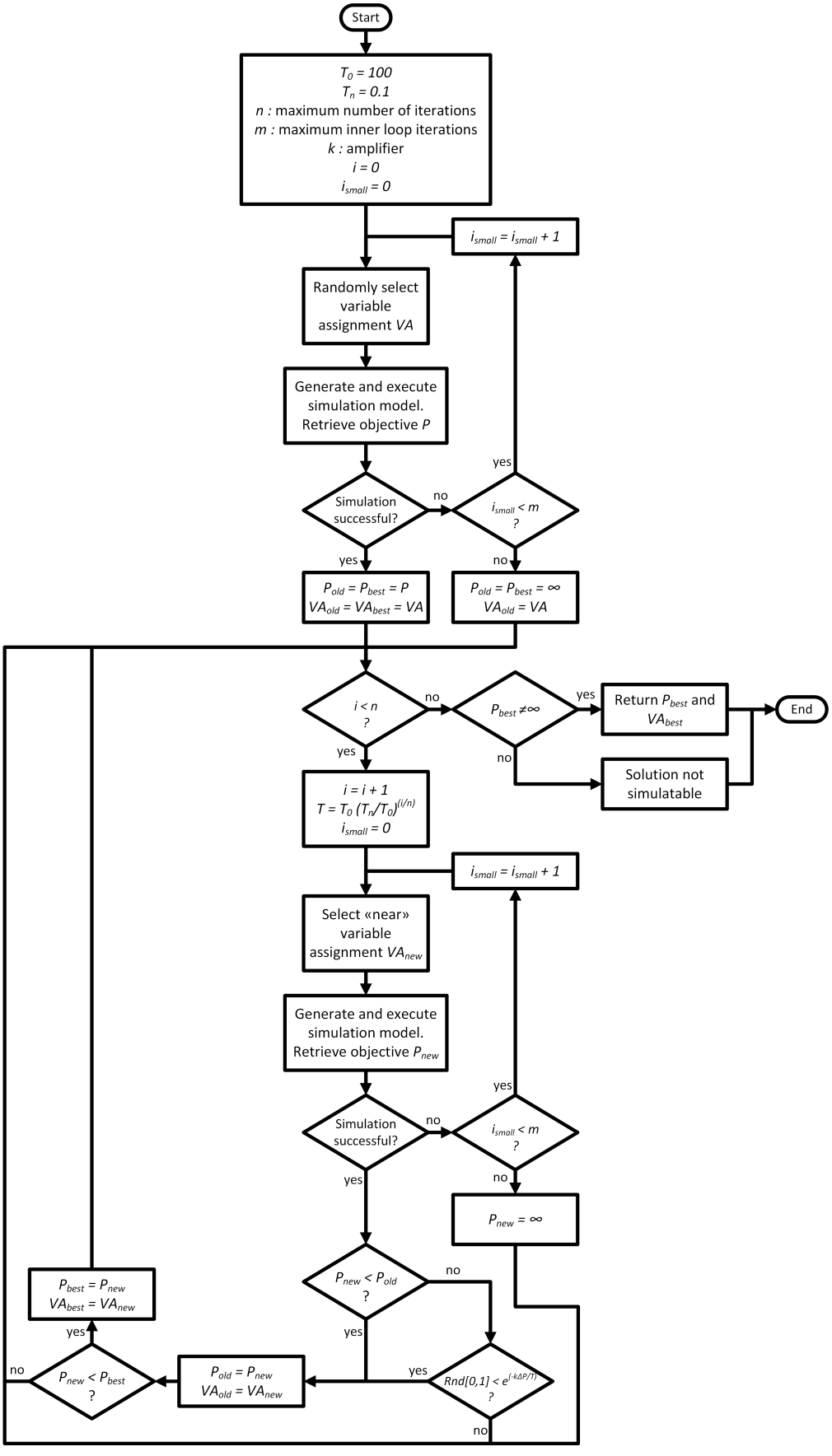


Figure 25: Flow diagram of the simulated annealing process. n, m, and k are defined by the user via the metamodel.

objective function value P is retrieved from the simulation results if possible.

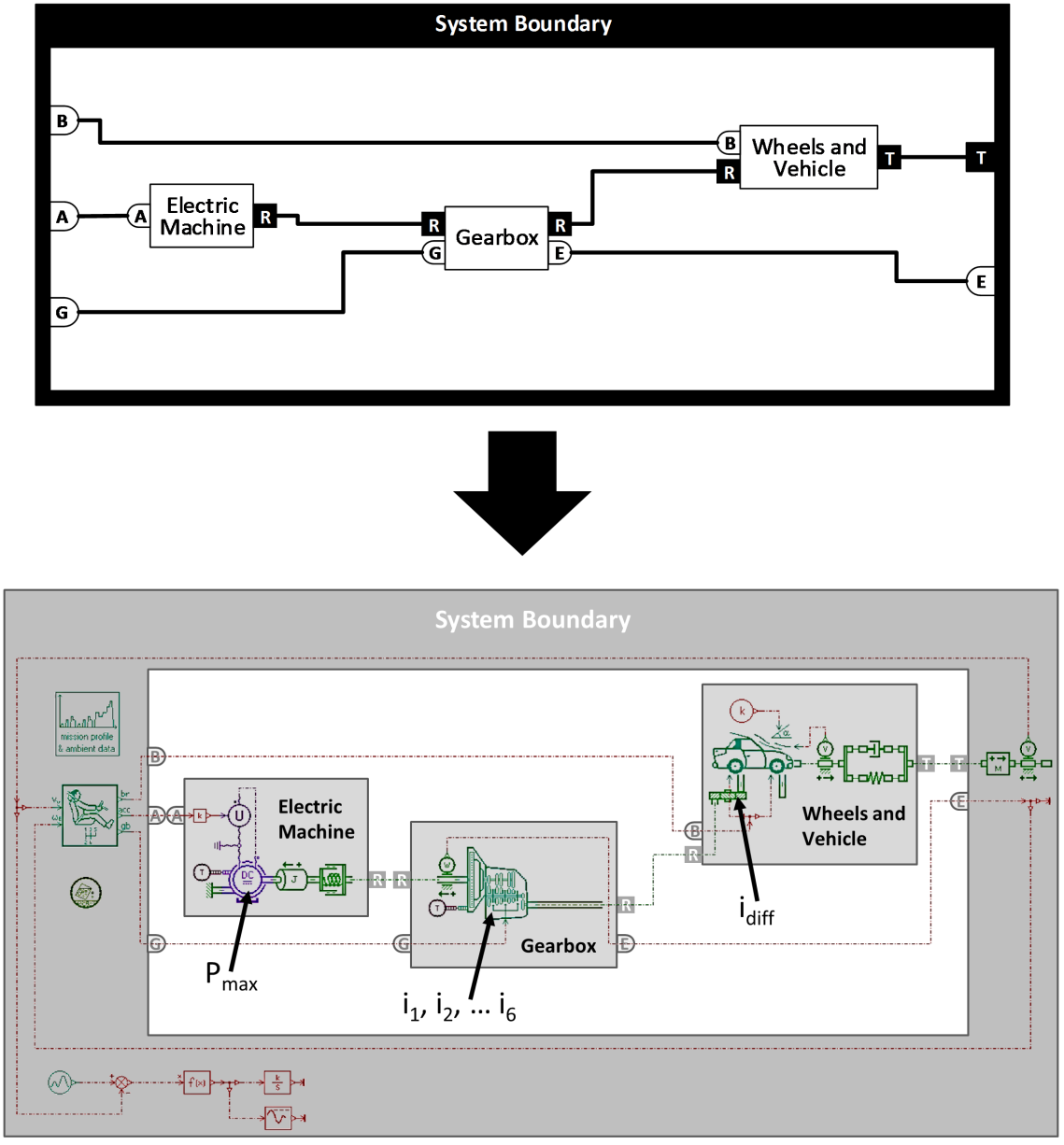


Figure 26: A CMG of an electric powertrain and its combined simulation model in LMS Amesim.

Disregarding a well-defined simulation model, the simulation might fail. This can happen, when partial simulation models are driven to states that were not intended and cannot be represented. An example for this is when the simulation model of a combustion engine, due to the other elements in the concept, is forced towards reversing the rotational speed at the flange. This state is not modeled and, therefore, the simulation aborts. If this happens, a new variable assignment is selected randomly and then simulated. This is done until either a variable assignment is found that would simulate without aborting or the number of iterations in this inner loop reaches r. If the simulation is successful, the performances P_{old} and the best performance found yet P_{best} is set to P. The same is

done for the variable assignment itself. In the case that no variable assignment is found that can be simulated, P_{old} and P_{best} are set to positive infinity and VA_{old} to VA. Here, VA_{best} remains undefined. P_{best} has to be defined, since in the end of every outer loop, the resulting P_{new} is compared to P_{best} and, once the end of the process ends, $P_{best} \neq \infty$ is checked to consider if a variable assignment was found, that could be simulated.

Beginning the outer loop, first it is checked if the abort criterion, i.e. the maximum number of iterations, has been reached. If not, i is increased, the temperature T_i for the current iteration is calculated using Eq. 56 and i_{small} is reset to zero. The temperature schedule, i.e. a vanilla schedule [91] that results for n iterations is shown in Figure 27, with T_0 being the starting temperature, T_n being the target temperature, and n being the maximum number of iterations. The vanilla schedule is defined as $T_i = \alpha T_0^i$ [91]. Here, α is set according to Eq. 57 to assure that the target temperature T_n is reached after n iterations.

$$T_i = T_0 \cdot \left(\frac{T_n}{T_0}\right)^{\frac{i}{n}} \tag{56}$$

$$\alpha = \left(\frac{T_n}{T_0}\right)^{\frac{1}{n}} \tag{57}$$

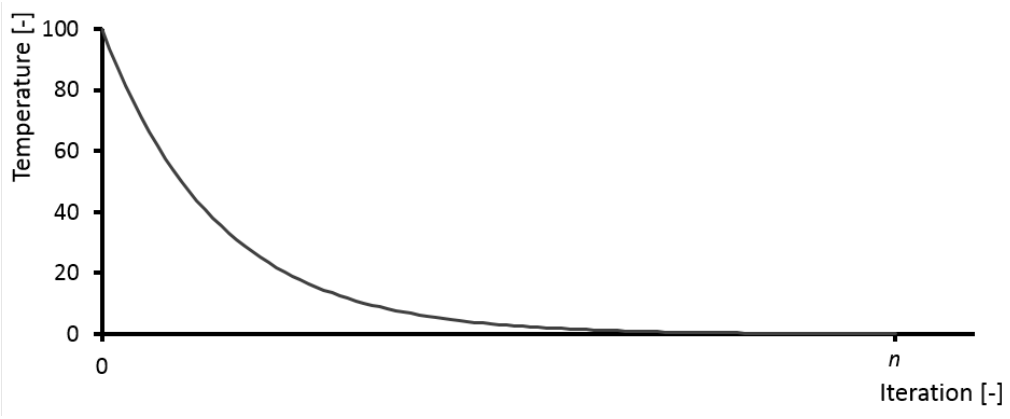


Figure 27: The temperature schedule that results from Eq. 56 and n iterations with $T_0 = 100$ and $T_n = 0.1$.

A variable assignment VA_{new} "near" VA_{old} is randomly selected from the set of variable assignments. To do this, the Euclidean distance between VA_{old} and all remaining variable assignment is calculated. All variable assignments that have a distance from VA_{old} that is smaller than 10% of the distance to the variable assignment that is the furthest away. As described before, the simulation model is generated and executed. If the simulation fails, as described above, an inner loop is started looking for another variable assignment that can be simulated near VA_{old}. Given the case that within r iterations in the inner loop no such variable assignment is found, the next iteration of the outer loop is started. Given that the simulation is successful, it is evaluated if the performance P_{new} of VA_{new} is better than P_{old}. If P_{new} is smaller than P_{old}, VA_{new} becomes the starting point for the next iteration

of the outer loop, by setting $P_{old} = P_{new}$ and $VA_{old} = VA_{new}$. If P_{new} is also smaller than P_{best}, the current variable assignment VA_{new} is the new VA_{best}. However, if P_{new} is larger than P_{old}, Eq. 58 is used to determine if VA_{new} should serve as a starting point for the next iteration of the outer loop nonetheless.

$$\text{Rnd}\,[0,1] < e^{-\frac{k \cdot \Delta P}{T}} \tag{58}$$

In Eq. 58, the Metropolis criterion (cf. Section 3.4) is enhanced by k as amplifier of $\Delta P = P_{old} - P_{new}$. The amplifier k has to be defined by the user in the system boundary and is used to tune the acceptance rate of variable assignments that lead to worse objective function values than their predecessors. In Figure 28 the influence of amplifier k on the acceptance of an objective function value with $\Delta P = 0.001$ is shown.

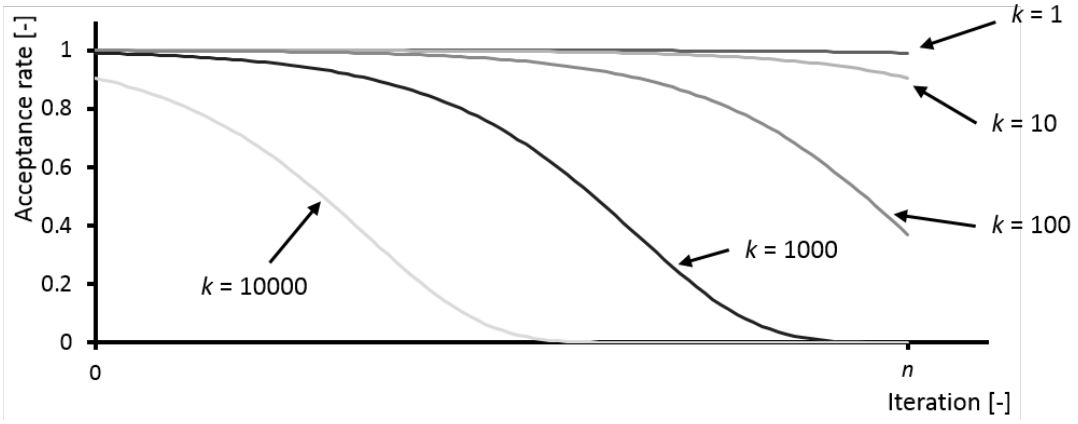


Figure 28: The acceptance rate of an objective function value with $\Delta P = 0.001$ for different settings of the amplifier k.

Once the maximum number of iterations for the outer loop is reached, it is checked if $P_{best} \neq \infty$. If yes, VA_{best} becomes the final variable assignment as well as P_{best} becomes the final performance. If $P_{best} \neq \infty$ is false, no variable assignment has been found within $n + (n \cdot r)$ different tested variable assignments. In this case, the solution is marked for further investigations by the designer.

4.5 Summary

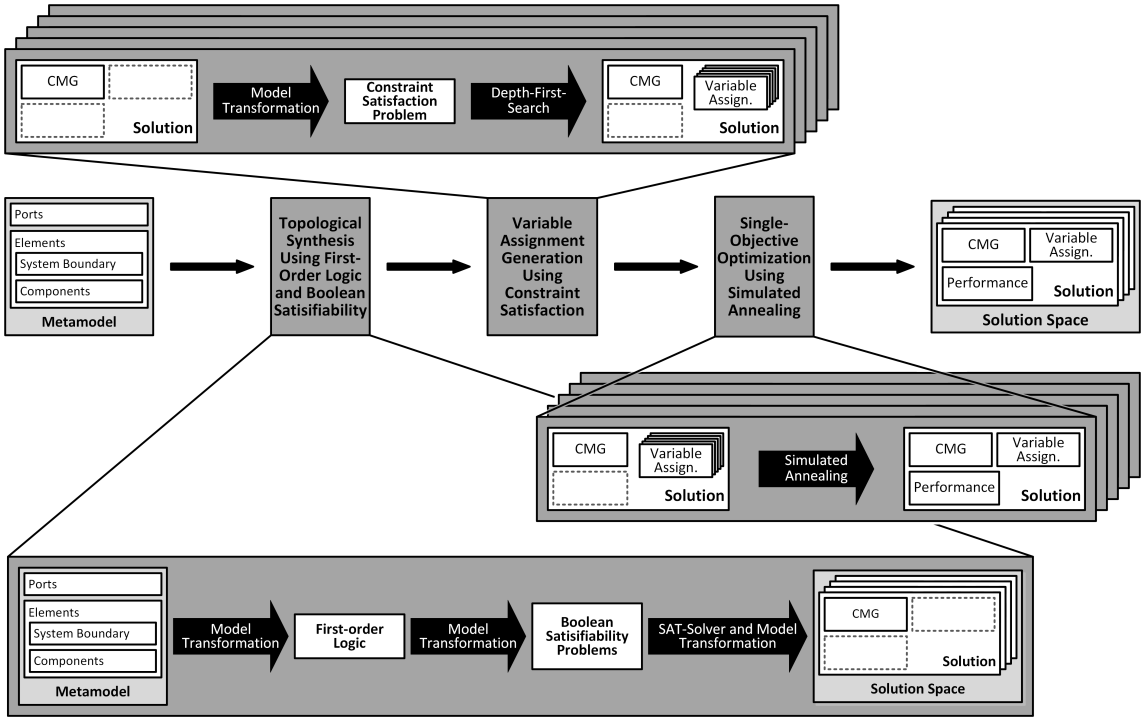


Figure 29: Complete view of all the different parts of the presented method.

Figure 29 shows the complete overview of the method with all its steps. Starting with the metamodel in the object-oriented graph-based environment, applying the topological synthesis using first-order logic and Boolean satisfiability, a solution space of topologies is created. For each of the generated solutions, if possible, variable assignments are determined that apply to the operation points and constraints in the variable assignment generation. Finally, optimization using simulated annealing attempts to find the variable assignment with the optimal performance regarding the objective function. In the end, a solution comprises a CMG with its optimal variable assignment and the associated performance, i.e. the simulation results.

This method introduces generic model transformations between the metamodel and a first-order logic description, between CMGs and constraint satisfaction problems, and between CMGs and bond graph-based simulation models. Using this transformations it presents an approach that can be fully automated after the definition of the metamodel.

5 Implementation

To show the capabilities of the method presented in Section 4 and to evaluate it, a software prototype is implemented. This section of the thesis describes the implementation. First, in Section 5.1 an overview of the software prototype is given. Section 5.2, then, presents descriptions of the tools that are used in the software prototype. After that, Section 5.3 shows how the metamodel definition is realized in the software prototype before Section 5.4 introduces the internal procedures that take place during the application of the prototype.

5.1 System overview

Figure 30 gives an overview of the implemented software prototype.

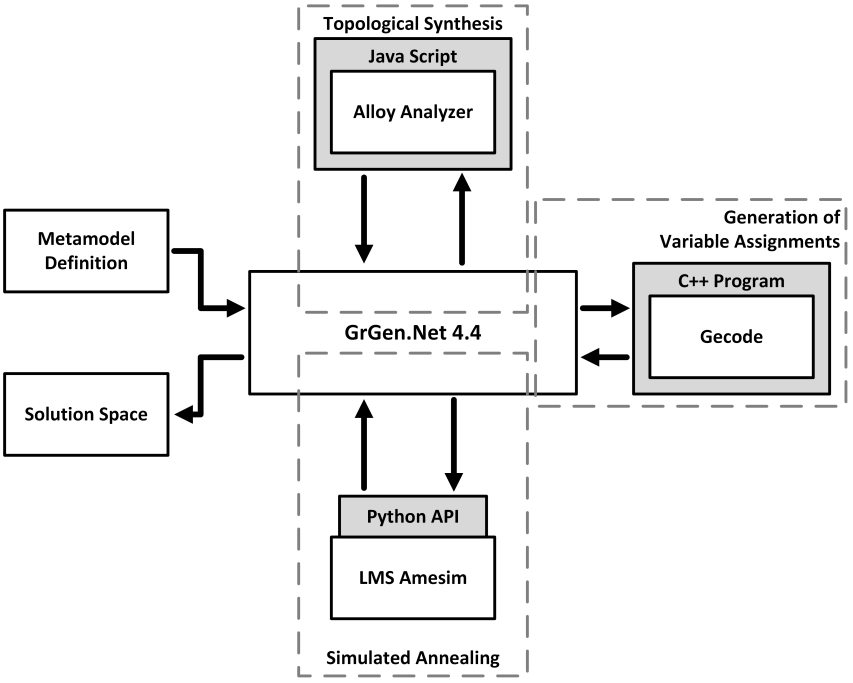


Figure 30: Schematic overview of the generic structure of the software prototype.

The framework of the prototype is GrGen.Net. Using its built-in C# interface, the synthesis process is controlled and different model transformations are carried out. Once the metamodel is defined, the first order logic models are created in C#. A java script is executed that accesses the API of Alloy analyzer. It loads the generated logics into Alloy and solves them internally using MiniSat, a solver for Boolean Satisfiability Problems. The resulting models to the first order logic are transformed into graphs in GrGen.Net. Following the topological synthesis, the generated CMGs are transformed into constraint satisfaction problems. A C++ program including the generated CSP is compiled. Using Gecode's libraries, the CSP is solved to generate the needed number of variable assignments for each solution. Finally, the parametric optimization for each generated solution

is carried out. To do this, different variable assignments for the simulation models created according to the CMGs are simulated. The python API of LMS Amesim is used to build and execute the simulation models before retrieving the simulation results.

5.2 Used tools

As mentioned above, different tools are used to implement the method. GrGen.Net (Section 5.2.1) is the framework for the implementation, the Alloy Analyzer (Section 5.2.2) offers a way to implement relational first-order logic descriptions, translate them to Boolean satisfiability problems and solve those using included SAT-solvers, Gecode (Section 5.2.3) is a toolkit to define and solve constraint satisfaction problems, and LMS Imagine.Lab Amesim (Section 5.2.4) is a commercial simulation suite. These tools are briefly described in the following sections.

5.2.1 GrGen.Net 4.4

GrGen.NET is a tool that offers languages for graph modeling and transformation [97]. Further, it provides a graph transformation engine. GrGen.NET defines three different languages:

- The *graph modeling language* allows the definition of metamodels based on edge types, node types and user defined types that are realized in C# classes. It enables multiple inheritance and the definition of attributes. In the software prototype implementation it is used to define the metamodel for the method.
- The *rule set language* offers the possibility to define graph transformation rules. Since the presented method is not using graph transformation rules, this language has no meaning for the prototype.
- The *rule application control language* offers a means to execute and combine graph transformation rules and user defined functions. Complex combination of rules and functions are realized using the Boolean return value of each.

Further, GrGen.Net provides the possibility to define individual functions and sequences using a built-in C# interface that can be executed. The main part of the prototype is realized within this interface in C#. Here solutions and CMGs are built, the model transformations are mostly carried out and the overall process is controlled.

5.2.2 Alloy Analyzer

The Alloy Analyzer is a front-end to the tool kodkod [98, 86]. It provides a programming language with the same name that has been developed to provide an easy way to program logical problems that are rooted in the field of relational and first-order logic. Having formulated such a problem, the Alloy Analyzer transcribes it into a kodkod suitable input. kodkod translates the logic into a suitable input for the SAT-Solver[11]. To solve

[11] For details on the transformation please refer to Section 3.1 and [98, 85]

this so-called clausal normal form of the Boolean satisfiability problem it is transferred to the SAT-solver. Different third-party SAT-solvers are part of the Alloy package: MiniSat [99], SAT4J [100], BerkMin [101] and zChaff [102]. For this implementation, the conflict-driven SAT-solver MiniSat [99] was used. MiniSat is a fast, complete and efficient solver. Here, "complete" means that the solver is able to iterate over all possible Boolean solutions. MiniSat checks the satisfiability of the transferred SAT-problem and returns, if the problem is satisfiable, a solution. kodkod now turns the Boolean solution into an *Alloy* model.

5.2.3 Gecode 4.4.0

Gecode [103] is a framework for the development of constraint based systems and applications. It is developed at KTH Stockholm. Gecode offers float, integer, and boolean variables and constructs to build constraints over these variables. To solve constraint satisfaction problems, Gecode offers two different search engines, depth-first search and branch-and-bound search that can be also used in restart-based search as meta-search engine. To allow the search engines to return to previous states, a hybrid ar recomputation and cloning is used. Gecode models are programmed in C++ linking to the needed Gecode headers and libraries.

5.2.4 LMS Imagine.Lab Amesim

Amesim[12] [104] is a commercial 1D simulation software that was originally developed by LMS International and is now continued by Siemens PLM Software. It supports multi-domain system modeling and analysis. In its core it is bond-graph based. Amesim provides extensive validated libraries for different physical domains, e.g. fluids, electronics, mechanics, thermodynamics, etc. and application libraries, for instance for air conditioning, automotive powertrains and internal combustion engines. Apart from a GUI[13], Amesim offers a C and a Python API to be accessed externally. Amesim offers fixed-step and variable-step integration algorithms to solve differential equations. There are three options for fixed-step integration methods: Euler, Adams-Bashforth, or Runge-Kutta. To run simulations most effectively using variable-step integration, Amesim employs the methods DASSL [105] if implicit variables are present and LSODA [106] otherwise. Both methods comprise a collection of different solvers and decide which one to use by monitoring the characteristics of the differential algebraic equations that are to be solved.

5.3 Metamodel Definition

To define the metamodel for the synthesis, the graph modeling language of GrGen.Net is used and extended. Element types are modeled as node types, port types as C# classes and the relation type as an undirected edge. Three subclasses inherit from the general port class. One with no variable that is transmitted, one with one variable, its type (integer

[12] from **A**dvanced **M**odeling **E**nvironment for performing **SIM**ulations)
[13] graphic user interface

or double) and minimum and maximum value, and one with two variables, their types, and minimum and maximum values. Based on these three types, the port hierarchy for the synthesis task is developed. Energy port types, for instance, pass a flow and an effort variable and, therefore, inherit from the port type with two variables. Signal port types that only pass one value inherit from the port type with one variable.

```
node class Element
{
    id:string;
    name:string;
    portdefstring:string;

    active_ports:array<Port>;
    inactive_ports:array<Port>;

    availability:boolean = false;

    supercomponent:string;
    submodel:string;
    submodel_dir:string;
    fixed_parameters:array<string>;
    variable_parameters:array<string>;
    plot_variables:map<int,string>;

    constants:map<string,string>;
    parameters_to_solve:map<string,string>;
    i_o_relations:array<string>;
    additional_constraints:array<string>;

    csp_sim_linker:map<string,string>;
}
```

Figure 31: Definition of the abstract element node type.

The definition of the basic element node type is shown in Figure 31. Every element contains the following attributes:

- `id` is an unique identification string which is set during the instantiation of the particular element.
- `name` is a string that contains the name of the element.
- `portdefstring` is the port definition string. Here, the user includes all the required information to define the port configuration of the element.
- `active_ports` is a list of Port objects that represent the obligatory ports in the element's port configuration. It is created during the instantiation of the element.

- `inactive_ports` is a list of Ports objects that represent the optional ports in the element's port configuration. It is created during the instantiation of the element.
- `availability` is a Boolean variable that is set by the user to indicate if the element is supposed to be available for the topological synthesis. Since the general element type is not supposed to be part of any solution this is set to `false`.
- `supercomponent` is a string containing the name of the supercomponent in Amesim that is associated to the element.
- `submodel` is a string containing the submodel of the elements supercomponent.
- `submodel_dir` is a string containing the submodels location on the harddrive.
- `fixed_parameters` is an array of strings that allows fixing certain parameters in the simulation model to constant values.
- `variable_parameters` is an array of strings that allows a parameter exploration within a simulation model. For the parameter a minimum and maximum value and a step size can be fixed to investigate the effects of a parameter.
- `plot_variables` is an array of strings that is set to define the resulting variables that are to be stored after the simulation.
- `constants` is a map of strings that is intended to include constant names and there values when such are needed in the definition of the additional constraints and input-output-relations.
- `parameters_to_solve` is a map of strings that allows defining variables for which an assignment is supposed to be found during the variable assignment generation.
- `i_o_relations` is an array of strings that include the relations between the in-ports and out-ports following the Minimodel convenience [107].
- `additional_constraints` is an array of strings that include the additional constraints for the variable assignment generation.
- `csp_sim_linker` is a map of strings that links the variable assignments to parameters in the simulation model.

The implementation of the metamodel allows the definition of variable port configurations. That means, that the number of ports can vary between different instantiations of one element type. The definition of variable port configurations, however, is only available if the topological synthesis is the only step of the method that is carried out, since the simulation models always have a fixed number of open connectors and definition of several simulation models for one element is not supported.

To define partial simulation models, Amesim's feature of supercomponents is used. A supercomponent is a modeling element that includes a part of a simulation model. For

each needed element type, a supercomponent is created with a submodel. The supercomponent is added to Amesims libraries and the submodel stored on the harddrive. Modeling the partial simulation models follows the guidelines presented in Section 4.1. However, modeling in Amesim, though bond graph-based, does not always follow the appearance of a bond graph. An example for this is the modeling of electrical circuits. Figure 32 shows a simple electric circuit diagram, as it would also be modeled in Amesim, on the left and the corresponding bond graph on the right. Since an electric circuit has to be grounded and closed which is not explicitly included in the bond graph, this has to be reflected in the modeling of partial simulation models for electric components. Therefore, the electrical ports in the metamodel have to map to two connectors in the partial simulation model, one for the outgoing and one for the return line, corresponding to a two-core cable.

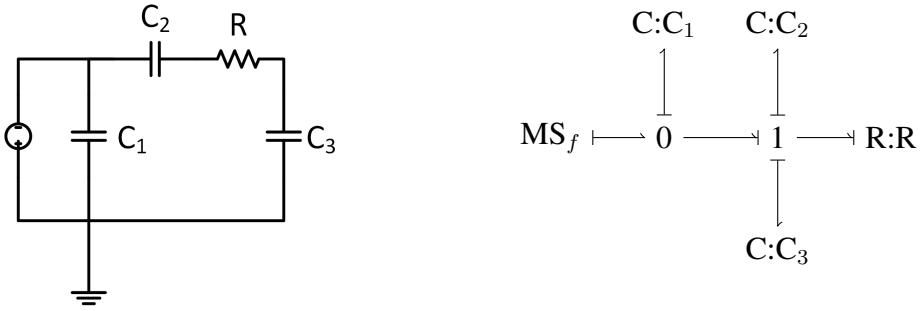


Figure 32: A simple electric circuit with corresponding bond graph, reproduced from [88]

For the system boundary, further, the objective function value is calculated in the partial simulation model. In order to suit to the software application, it has to be defined that the intended value is passed at the final time-step of the simulation.

In Figure 33, the definition of the gearbox component is presented. Here, examples for the different attributes are shown. The port definition string defines four different ports, the first one is defined in the following statement:

```
[Rotational;in;1;1;[-160.0,160.0];[-50.0,50.0];[InShaft];[3]]
```

The different parts of this statement mean the following:

- `Rotational` refers to the type of the port, here rotational mechanical energy
- `in` refers to the direction of the port, here it is an in-port
- The first `1` refers to the minimum number of this port in the port configuration
- The second `1` refers to the maximum number of this port in the port configuration, so this component has one rotational mechanical in-port
- `[-160.0,160.0]` describes the flow range of this port in SI units. For this gearbox it is assumed that it can run in both senses of rotation between -160 $\frac{rad}{s}$ and 160 $\frac{\text{rad}}{\text{s}}$.

```
node class Gearbox extends Element
{
    name = "Gearbox";
    portdefstring = "[Rotational;in;1;1;[-1600.0,1600.0];[-500.0,500.0];[InShaft];[3]]
                     [Rotational;out;1;1;[-2700.0,2700.0];[-2500.0,2500.0];[OutShaft];[1]]
                     [Gear;in;1;1;[Gear];[0];integer:0:5]
                     [ESpeed;out;1;1;[EngineSpeed];[2];float]";
    availability = true;

    supercomponent = "Gearbox";
    submodel = "GEARBOX_CDS_1";
    submodel_dir = "D:\\AMSimLib\\Metamodel\\submodels";
    fixed_parameters = array<string>["paratio@drv_MGBwithClutch2,1.00"];

    parameters_to_solve = map<string,string>{   "gear" -> "6, 0.6, 5.0" };
    constants = map<string,string>{"one" -> "1.0"};
    i_o_relations = array<string>[  "((InShaft_flow == gear[Gear_sig] * OutShaft_flow) [...]"
                                    "EngineSpeed_sig == InShaft_flow"   ];

    additional_constraints = array<string>[ "gear[0] > gear[1]",
                                            "gear[1] > gear[2]",
                                            "gear[2] > gear[3]",
                                            "gear[3] > gear[4]",
                                            "gear[4] > gear[5]" ];

    csp_sim_linker = map<string,string>{    "gear[0]" -> "trratio1@drv_MGBwithClutch2",
                                            "gear[1]" -> "trratio2@drv_MGBwithClutch2",
                                            "gear[2]" -> "trratio3@drv_MGBwithClutch2",
                                            "gear[3]" -> "trratio4@drv_MGBwithClutch2",
                                            "gear[4]" -> "trratio5@drv_MGBwithClutch2",
                                            "gear[5]" -> "trratio6@drv_MGBwithClutch2"  };
}
```

Figure 33: Definition of the gearbox element type.

- `[-50.0,50.0]` describes the effort range of this port in SI units. For this gearbox it is between -50 Nm and 50 Nm.

- `[InShaft]` is the label of the port, to which is referred in the definition of input-output-relations or operation points. If the port configuration contains more than one of this port, the different labels for the instances are given in the square brackets separated by commas.

- The expression `[3]` corresponds to the identifier of the open connector in the partial simulation model in Amesim. If the port configuration contains more than one of this port, the different identifier for the instances are given in the square brackets separated by commas to link all the port instances to individual open connectors.

As can be seen in Figure 34, the system boundary contains, in addition to the already mentioned, additional attributes:

- `optimization_variables` contains the name of the result variable that delivers the objective function value for the optimization. It is designed as an array to support multi-objective optimization which should be considered in future work (cf. Section 7).

- `OP_setting` is an array of strings that include the operation points as flow-, effort-, and signal values for the specific ports that are identified using their labels.
- `run_parameters` is an array of strings which define the run parameters for the simulation, i.e. start time, stop time, write interval etc.
- `maximum_number_of_components` defines the maximum number of components to be used in the topological synthesis.
- `number_of_variable_assignments` defines the number of variable assignments to be created for each solution.
- `optimization_amplifier` refers to the amplifier k (cf. Eq. 58) that is used to control the acceptance rate during simulated annealing.

```
node class System_Boundary extends Element
{
    name = "System_Boundary";
    portdefstring =
    "[Translational;in;1;1;[0.0,100.0];[0.0,2000.0];[Road1];[3]]
    [Acceleration;out;1;1;[Acc];[1];float]
    [Brake;out;1;1;[Brake];[2];float]
    [Gear;out;1;1;[Gear];[0];integer:0:5]
    [ESpeed;in;1;1;[EngineSpeed];[4];float]";
    availability = true;

    supercomponent = "Problem_3";
    submodel = "CDS_P3_AUS";
    submodel_dir = "D:\\AMSimLib\\Metamodel\\submodels";
    plot_variables = map<int,string>{ 3->"[vcont@drv_driver_a2,vveh@drv_driver_a2,output@elect02,output@peaktopeak]" };
    optimization_variables:array<string> = ["output@elect02"];

    maximum_number_of_components:int = 10;
    number_of_variable_assignments:int = 10000;
    optimization_amplifier = 1;

    run_parameters:array<string> = ["start_time_s,580.0","stop_time_s,1180.0","interval_s,0.25"];

    OP_setting:array<string> = [    "0;Road1;0.0;m/s;0.0;N",        "0;Gear;0;int",     "0;Break;1;float",  "0;Acc;0;float",
                                    // NEDC - UDC
                                    "1;Road1;2.08;m/s;1879.77;N",   "1;Gear;0;int",     "1;Brake;0;float",  "1;Acc;1;float",
                                    "2;Road1;4.17;m/s;169.48;N",    "2;Gear;0;int",     "2;Brake;0;float",  "2;Acc;0;float",
                                    "3;Road1;6.53;m/s;1731.55;N",   "3;Gear;1;int",     "3;Brake;0;float",  "3;Acc;1;float",
                                    "4;Road1;8.89;m/s;196.52;N",    "4;Gear;1;int",     "4;Brake;0;float",  "4;Acc;0;float",
                                    "5;Road1;11.67;m/s;1079.56;N",  "5;Gear;2;int",     "5;Brake;0;float",  "5;Acc;1;float",
                                    "6;Road1;13.89;m/s;246.46;N",   "6;Gear;2;int",     "6;Brake;0;float",  "6;Acc;0;float",
                                    // NEDC - EUDC
                                    "7;Road1;16.67;m/s;1422.19;N",  "7;Gear;3;int",     "7;Brake;0;float",  "7;Acc;1;float",
                                    "8;Road1;19.44;m/s;327.68;N",   "8;Gear;3;int",     "8;Brake;0;float",  "8;Acc;0;float",
                                    "9;Road1;23.61;m/s;802.35;N",   "9;Gear;4;int",     "9;Brake;0;float",  "9;Acc;1;float",
                                    "10;Road1;27.78;m/s;500.26;N",  "10;Gear;4;int",    "10;Brake;0;float", "10;Acc;0;float",
                                    "11;Road1;30.56;m/s;1033.32;N", "11;Gear;5;int",    "11;Brake;0;float", "11;Acc;1;float",
                                    "12;Road1;33.33;m/s;649.15;N",  "12;Gear;5;int",    "12;Brake;0;float", "12;Acc;0;float"];
}
```

Figure 34: Definition of the system boundary element type.

5.4 Application

The application begins and ends in GrGen.Net. Here, the different processes are triggered and the results are presented. In order to be able to make use of the extended information in the metamodel, functions are implemented to handle the initialization of solutions and to build and include a CMG within them. In this section, first, a more detailed view on the path from the metamodel to a set of solutions with CMGs is given before the generation

of variable assignments and the variable optimization are presented. The focus lies on the communication between the different tools and their integration in the process.

5.4.1 Topological Synthesis

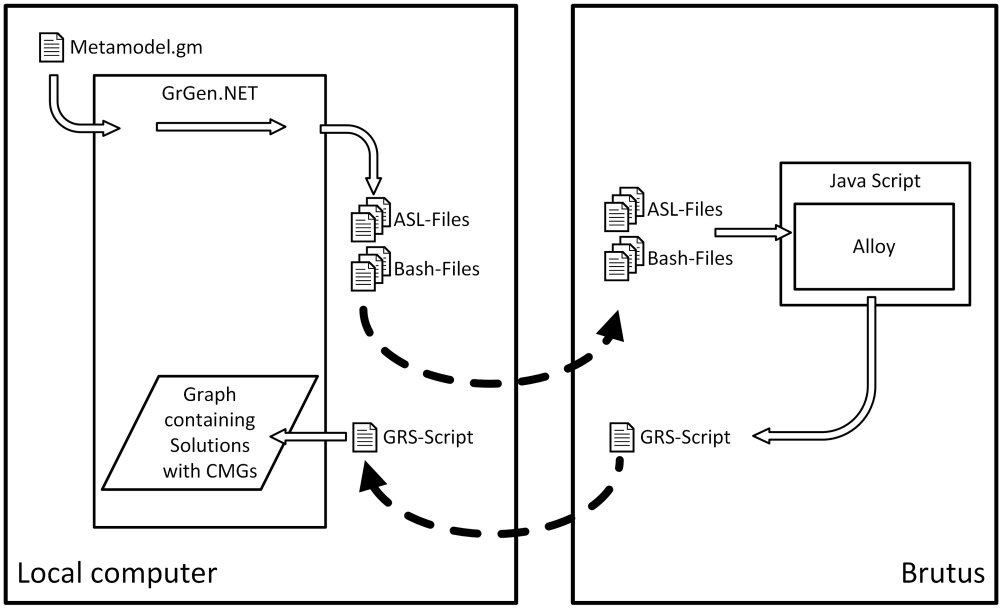


Figure 35: Topological synthesis implementation.

Figure 35 illustrates the implementation of the topological synthesis. The metamodel is analyzed and logical statements are created as described in Section 4.2. For each of the different scopes that are calculated an ASL-file is created. These ASL-files include the complete first-order logic written in Alloy. They each finish with an execution command that triggers the generation of solutions. The solutions for each individual scope have to exactly consist of the precise number of elements from the categories and ports as prescribed in the scope.

In order to cope with the expensive solving of the Boolean satisfiability problems, the translation of first-order logics to Boolean satisfiability problems and their solving is designed to be run on Brutus, the high performance cluster of ETH Zurich[14]. There, the different scopes are solved in parallel. Additional to the ASL-files, bash scripts are created to enqueue the needed calculations on Brutus. A java script has been written that accesses the Alloy Analyzer java-API, loads the ASL-file and starts the model generation using MiniSat. When a solution is found, it is analyzed and a GRS-file is created that generates the solutions including their CMGs in GrGen.Net. Therefore, C# functions are called that instantiate elements, add them to a graph and connect them according to the model generated in Alloy Analyzer.

As a means to support symmetry breaking, the prevention of redundant solutions due to multiple Boolean solutions mapping to one graph solution, first the string sets of that

[14]http://www.cluster.ethz.ch/index_EN

generate the different graphs are compared. If the strings are identical, the solution is disregarded. Solutions with different strings are included in the GRS-file. However, after each solution is generated, it is compared to the already generated solutions using GrGen.NET's graph isomorphism check. As a result, redundant solutions in the solution space are prohibited.

5.4.2 Variable Assignment Generation

To realize the generation of alternative variable assignments, the capabilities of Gecode are taken advantage of. A C++ application was developed that includes the Gecode libraries. Additionally, a header is included that contains the constraint satisfaction problem. In this program the constraint satisfaction problem is solved using the built-in restart-based depth-first search and random branching for the variables that need assignment.

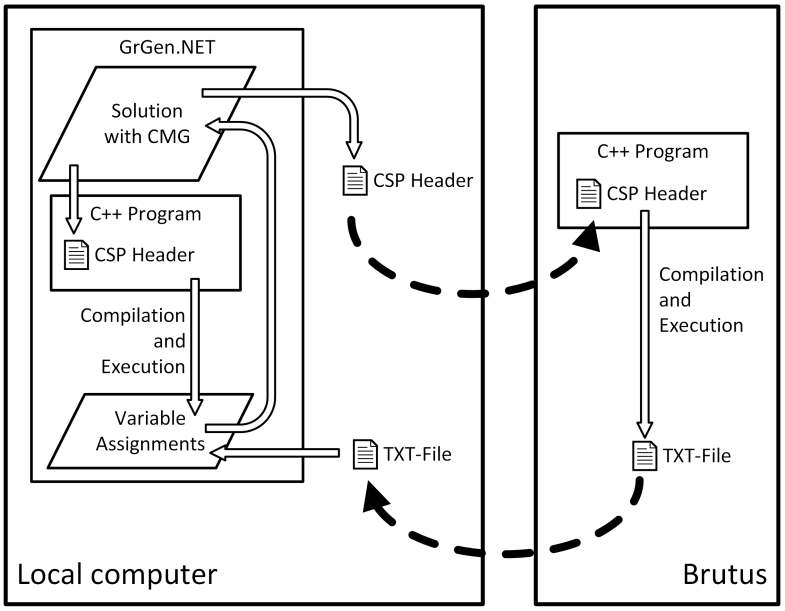


Figure 36: Variable assignment generation implementation.

As shown in Figure 36, the header-file containing the constraint satisfaction problem is generated from the CMG within each solution. The process resembles the method presented in Section 4.3. A process is invoked in GrGen.Net in which the C++ application is compiled and then executed. The output of the system, the variable assignments, are written to the standard output of the process which is read by the hosting process. The variable assignment are collected, and compared to each other to avoid redundant assignments which could result from the restarts and on the different number of decimal places between the different frameworks.

In the case of complex CMGs and rising numbers of operation points, the solving of the CSPs may become inefficient. To deal with this case, the functionality was included to generate the C++ header file without compiling and solving the actual program. Here

it is advantageous to use the parallel computing capabilities of Gecode and the possibility to run several instances of the program at the same time. Therefore, a second version of the C++ program was developed that runs on Brutus, the high performance cluster of ETH Zurich. The user copies the C++ header to Brutus, compiles the software there and runs the variable assignment generation there, adapted to his or her account limitations, i.e. the maximum number of cores that are allowed to be used. Running the application on Brutus, a txt-file is generated. This file is updated during the program application to be able to make also use of partial results. The txt-file is then copied back to the working directory of GrGen.Net where it is recognized for the variable optimization.

5.4.3 Variable Optimization

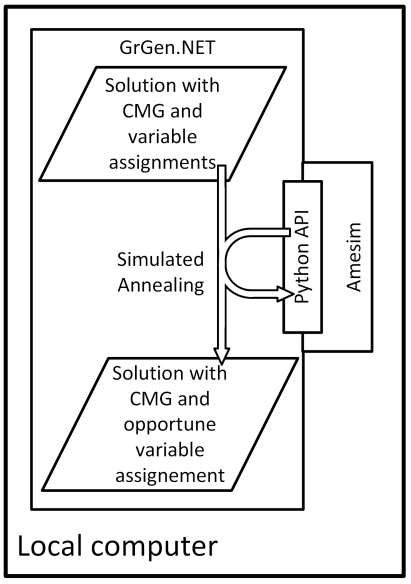


Figure 37: Variable optimization implementation.

In the variable optimization, simulation models are generated from the different CMGs. As illustrated in Figure 37, the CMG is analyzed and a python script is derived that includes the right supercomponents, their particular submodels, their variables and the connections in between them. Further, a routine to write the result variables that are to be available in the CMG to the console is included in this script. Amesim includes a Python console which is started from GrGen.Net, the script is transmitted to the console and the results are read from there. The results are then stored within the CMG and can optionally be saved as CSV[15]-file.

5.5 Summary

In this section, the software prototype is described. The focus lies on how to define a metamodel that is applicable for the presented method. Further, the tools that were used in the

[15] comma separated value

implementation are described and how they communicate. In contrast to the method, the implementation only realizes a semi-automated work-flow. Due to the batch system that works on Brutus, directly accessing the calculation nodes on the cluster is not possible. The files that are needed to run the calculations on Brutus are generated, but the user has to transfer them to the cluster and queue them in the batch system manually. The software prototype is a means to show the potential of the method on four case studies, which are presented in the following section.

6 Case Studies

In this section, four case studies are presented. The goal of this chapter is to provide a proof-of-concept for the presented method and to show its suitability for supporting conceptual design for technical systems of different order of complexity. Four orders of complexity, ranging from parts to plants are defined by Hubka and Eder [93] and characterized in Table 2.

Table 2: Classification of technical system by degree of complexity (reproduced from [93])

Level of Complexity	**Technical System**	**Characteristic**	**Examples**
I (simplest)	Part, Component	Elementary system produced without assembly operations	Bolt, bearing sleeve, spring, washer
II	Group, Mechanism, Sub-assembly	System that can fulfill some higher functions	Gear box, hydraulic drive, spindle head, brake unit, shaft coupling
III	Machine, Apparatus, Device	System that consists of sub-assemblies and parts that perform a closed function	Lathe, motor vehicle, electric motor
IV	Plant, Equipment, Complex machine unit	Complicated system that fulfills a number of functions and that consists of machines, groups and parts that constitute a functional and spatial unity	Hardening plant, machining transfer line, factory equipment

The first two cases studies are preliminary cases studies that are conducted using an earlier software prototype where only the topological synthesis using Boolean satisfiability was implemented. The scope calculation in this prototype only distinguished the different scopes using the total number of elements and ports in the solution. Further, this prototype used the java-based SAT-Solver SAT4J [100].

The first case study (Section 6.1) shows an application of the topological synthesis to higher-order technical systems, i.e. the synthesis of a chemical plant. Different possibilities to realize a chemical process, i.e. the ethylene to ethyl alcohol process, are generated. This case study is conducted to show the applicability of the topological synthesis to technical systems of a high level of complexity (IV) [93] and outside the domain of energy- and signal-based systems. The results are validated against literature in chemical process design [108].

The second case study (Section 6.2) is a preliminary study tackling the generation of automotive powertrain topologies. Here, the applicability of the topological synthesis for

technical systems of lower level of complexity (III) [93] is presented. The cases study on automotive powertrains that was conducted by Helms and Shea [27] is implemented using this work's method and the results are compared.

Case study three and four are conducted with the final prototype as presented in Section 5. Here, in addition to the topological synthesis, also the variable assignment generation using constraint satisfaction and the optimization via simulated annealing are preformed.

Case study three (Section 6.3), provides another viewpoint to the case study about automotive powertrains. Here it is shown that technical systems of level of complexity III can not only be generated but also automatically simulated.

Finally, case study four (Section 6.4) presents the concept generation and optimization of printhead drives for a 3D printer. These sub-assemblies of 3D printers (level of complexity II [93]) consist of different basic kinematic mechanisms and linear drives. The successful integration of different domains (electric, mechanic, and hydraulic) and the feasibility of simulations with more than one dimension.

Section 6.5 concludes the findings from the case studies.

Each study is structured as follows: After a general introduction that puts the study in context and highlights the rationale behind it, the metamodel for the synthesis is presented. Following this, the metamodel of the particular case study is presented giving details to the components and the system boundary. Then, details from the application of the method are given, distributed by its different phases. Each case study description is concluded with the presentation and discussion of the results.

6.1 Case Study I - Chemical Process Engineering

This example comes from the domain of chemical process engineering; the case-study of the ethylene to ethyl alcohol process from [108] is used. The task can be described as follows: Water (W) and a feed of 95 mole-% ethylene (EL), 3% propylene (PL) and 1% methane (M) is to be converted into other products that can be sold. The desired products are ethylalcohol (EA) and diethylether (DEE). In general, the task is to design a chemical plant at a high level of abstraction. This case study's intent is to deliver a proof-of-concept of the topological synthesis and to show its suitability to tackle design task with complexity level IV (cf. Table 2). Further, it shows the method's capabilities outside the energy- and signal-based domain. The following sections give details about the metamodel, the method application and the results.

6.1.1 Metamodel

From the description in [108], a port hierarchy is developed that includes the chemical flows that are relevant for this process (see Fig. 38). The chemical flows are distributed

in four different categories: i) the desired products of the plant, ii) stable mixtures of by-products, iii) by-products that can be recycled in the process, and iv) byproduct considered waste. The different components and the system boundary is shown in Figure 39. Three chemical reactors and five different separators are modeled. The system boundary is modeled with the ports to the environment: W and EL, PL, M-mixture out-ports, and EA and DEE in-ports.

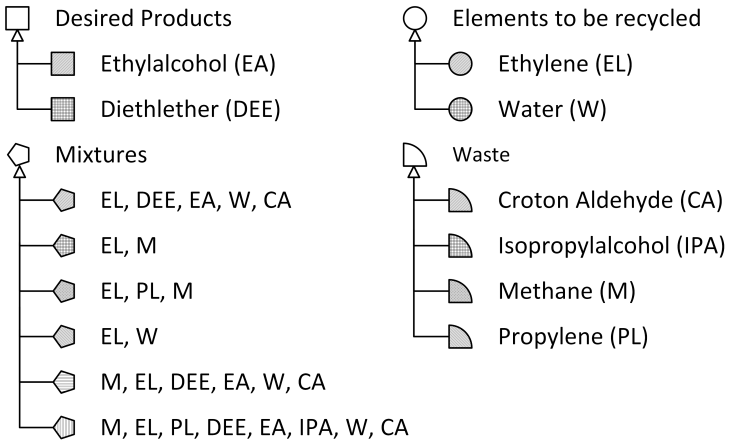


Figure 38: Port hierarchy for chemical process synthesis.

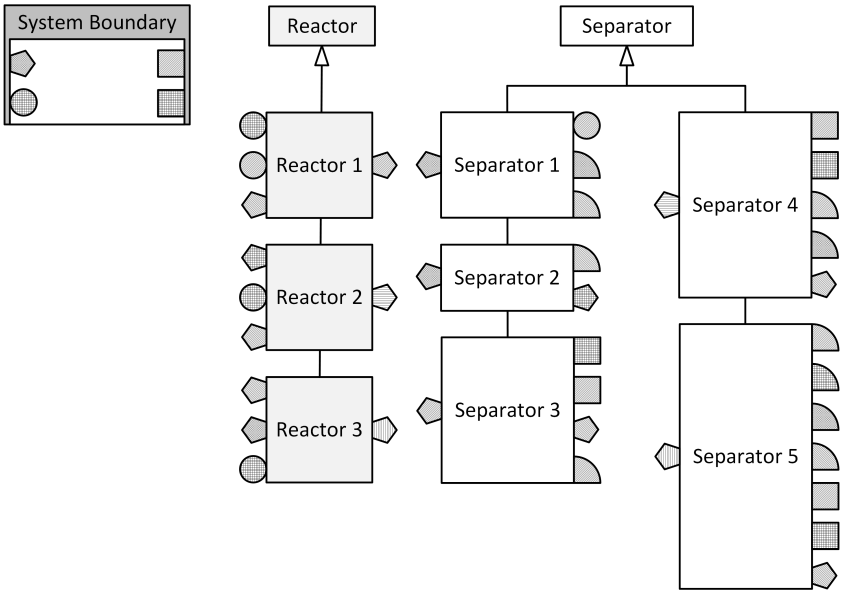


Figure 39: Element hierarchy for chemical process synthesis.

6.1.2 Method Application

In order to solve the design task that is modeled in the system boundary, the metamodel is reduced to those port types that are not considered waste. This reduced metamodel is

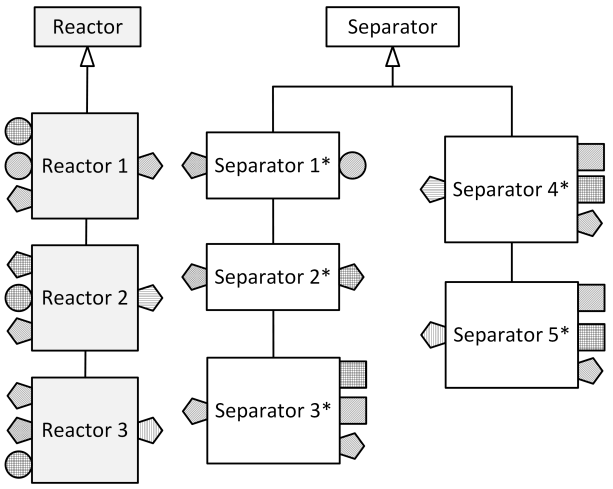


Figure 40: Reduced metamodel for the topological synthesis.

shown in Figure 40. The scopes are calculated as presented in Section 4.2.2. There are three different categories of elements:

- category 1 having one in-port and one out-port,
- category 2 having one in-port and three out-ports, and
- category 3 having three in-ports and one out-port.

In this case study, the environment provides two in- and two out-ports. Using the categories described above and five as maximum number of components, the scopes that are to be investigated for solutions are calculated. As mentioned before, this case study was conducted with a preliminary prototype, so the scopes are written as "number of elements/number of ports" instead of separated enumerators for every category. The scopes are composed of either

- no element at all (Scope: 1/4), or
- an arbitrary number of category 1 elements (scopes: 2/6, 3/8, 4/10, 5/12, 6/15), or
- one category 2 and one category 3 element with none to infinite elements of category 1 (scopes: 3/12, 4/14, 5/16, 6/18), or
- two combinations of one category 2 and one category 3 element with none to infinite elements of category 1 (scopes: 5/20, 6/22).

Building on these scopes the Boolean satisfiability problem is created and solved.

6.1.3 Results

Solving the process design task reveals all the different alternatives that are also found in [108]. One solution is found for scope 3/12 and two are found for scope 4/24. Samples in higher scopes lead to no further results. The three solutions are shown in Figure 41 and the same as the proposed solutions in [108]. The solutions in Figure 41 are represented

using the metamodel defined in Figure 39 which leads to the open ports. Those ports were not represented in the reduced metamodel in Figure 40 and, consecutively, during synthesis.

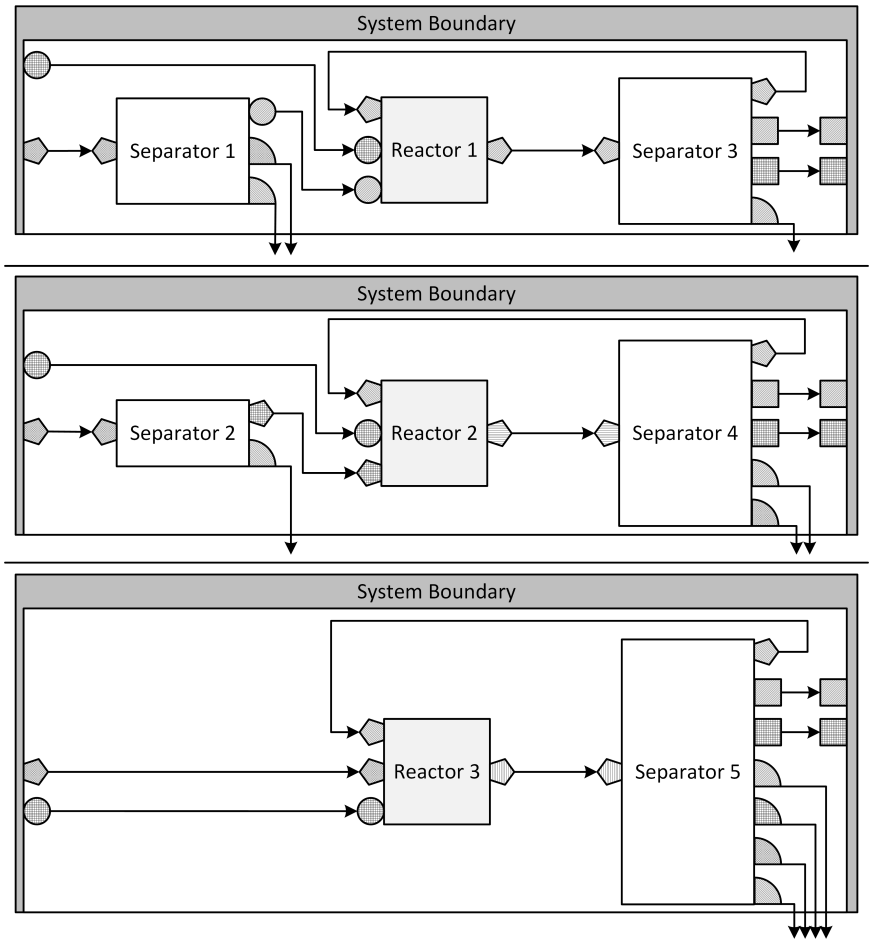


Figure 41: The three generated process alternatives for the chemical process synthesis task.

This example shows the capability of the topological synthesis for concept generation for technical systems with a high level of complexity. This is due to the power of the possibility to freely define port types. However, the modeling in this case study was carried out at a high level of abstraction and no evaluation of the generated concepts is available since it is outside the energy- and signal-based applications. Nevertheless, this study shows that the topological synthesis presented in this work is not only applicable to concept topology generation but also process synthesis.

6.2 Case Study II - Automotive Powertrains I

This case study is about the generation of automotive powertrain topologies, i.e. a technical system with level of complexity III (cf. Table 2). In [27], Helms and Shea present an approach to topological synthesis using object-oriented graph grammars. They tackled a problem of synthesizing concept topologies for automotive powertrains. Since, with their approach, only topologies with at most one branching point can be generated, this case study shows that since this work's method is able to also generate network-like structures, more solutions can be generated.

The case study in [27] is described as follows: Powertrain topologies are to be generated for four different, so-called, high-level functions:

1. "Displace vehicle electrically", i.e. the conversion of a signal- and an electric energy-flow into a mechanical translational energy-flow,
2. "Displace vehicle conventionally", i.e. the conversion of a signal- and a chemical energy-flow into a mechanical translational energy-flow,
3. "Recuperate energy electrically", i.e. the conversion of a mechanical translational energy-flow into an electric energy-flow, and
4. "Boost", i.e. the conversion of a electrical energy- and a chemical energy-flow into a mechanical translational energy-flow.

The metamodel for the generation in [27] consists of nine elementary functions, twelve physical principles/laws, and seven components all modeled as available elements for the topology as shown in Figure 42.

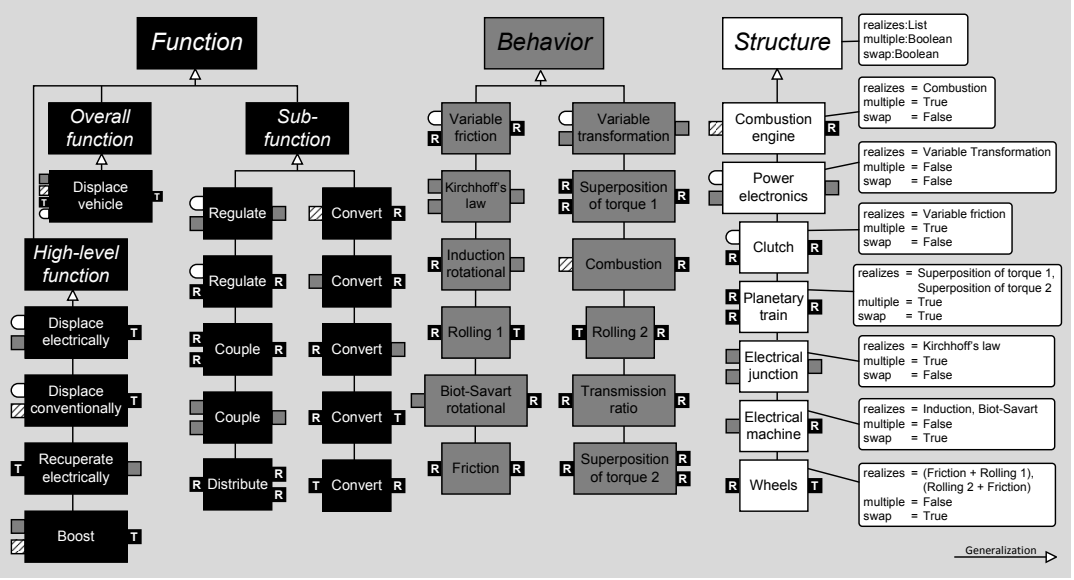


Figure 42: Metamodel for the approach of Helms and Shea [27]

The remainder of this section contains the description of the metamodel for this work's approach, and details for the method application. In the end, the results from the method application are presented and compared to [27].

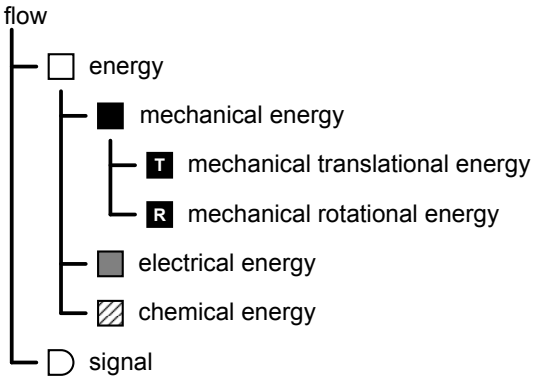


Figure 43: Port hierarchy showing different types of flow values.

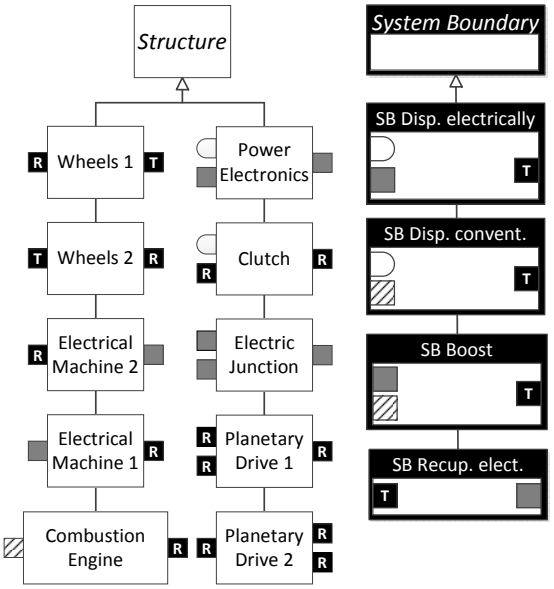


Figure 44: Metamodel for this work.

6.2.1 Metamodel

As presented in Section 2.1.3 on page 13, the approach of Helms and Shea [27] directly uses representations of the high-level functions as initial conditions for the synthesis. This work's approach, however, is based on PMPs. So, elements representing the system boundaries of the desired product have to be introduced that resemble the functions' port configurations but with swapped direction attributes. For example, the "Displace electrically" function, which has a signal and an electrical energy in-port, and a translational mechanical energy out-port, is represented by an system boundary that provides a signal and an electrical energy out-port, and a translational mechanical energy in-port.

The port hierarchy for this approach is shown in Figure 43. Figure 44 presents the metamodel. Since this work's method does not support the possibility of swappable components (attribute *swap* in Figure 42), such components are each represented twice, with mirrored port configurations. Therefore, ten different components result, as shown in Figure 44.

6.2.2 Method Application

For the scope calculation, three different component categories exist as shown in Figure 45:

- category 1: one in-port, one out-port
- category 2: two in-ports, one out-port
- category 3: one in-port, two out-ports

The system boundary has either two in- and one out-ports ("SB Displace electrically", "SB Displace conventionally", "SB Boost") or one in- and one out-port ("SB Recuperate

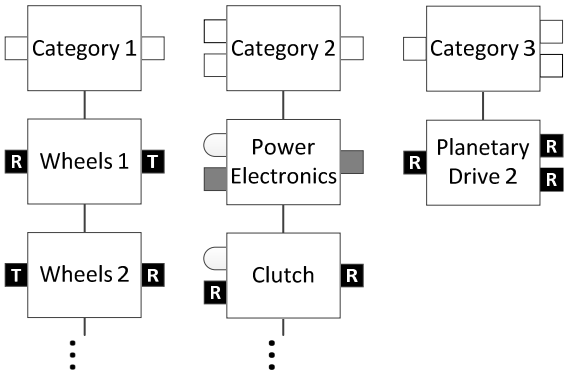


Figure 45: The elements structured in categories over their abstracted port configuration from Fig. 44

electrically"). Since no port in the resulting graph is allowed to be unconnected, only particular sets of elements are feasible. In order to match the port configuration of the two in-, one out-port problem element, there must always be one more category 2 element than category 3 elements. The number of category 1 elements is irrelevant. So, for this problem element the only valid combination can also be written as follows:

- n category 1 elements with $0 \leq n < \infty$,
- m category 2 elements with $1 \leq m < \infty$, and
- $m - 1$ category 3 elements.

The scope[16] can now be calculated using the absolute number of elements and ports for the specific categories:

- problem: 1 element, 3 ports
- category 1: 1 element, 2 ports
- category 2: 1 element, 3 ports
- category 3: 1 element, 3 ports

So, for this defined problem element, Eqs. 59 and 60 are used to calculate the number of elements z_e and number of ports z_p. The "$+1$" in Eq. 59 and the "$+3$" in Eq. 60 both at the end represent the system boundary.

$$z_e = m + (m - 1) + n + 1 = 2m + n \tag{59}$$

$$z_p = 3m + 3(m - 1) + 2n + 3 = 6m + 2n \tag{60}$$

[16]It has to be remarked that this cases study was conducted with an early software prototype that did not include the scope calculation as presented in Section 4.2.2

The scopes that result are as follows (written as z_e/z_p): 2/6, 3/8, 4/10, 5/12, etc. For the "Recuperate electrically" function the resulting scopes are: 2/4, 3/6, 3/8, 4/8, 4/10, 5/10, etc.

Scopes up to a maximum z_e of ten (corresponds to ten components in the solution and one system boundary) are solved using the SAT-solver SAT4J. The time needed to calculate the first solution for the *SB Boost* system boundary are presented in Table 3. As can be seen, the calculation time grows exponentially with the scope.

Table 3: Calculation times for the first solution – “Boost”-system boundary – Case Study I

Scope:	5/12	6/14	7/18	9/22	11/30
Time:	4 sec	7 sec	43 sec	1 min	28 min

6.2.3 Results

Table 4: Results

		Number of unique solutions							
Function	Scope (z_e/z_p):	3/6	4/10	5/12	6/14	6/16	7/18	8/20	8/22	9/22
Displace conventionally		-	1	-	2	-	-	-	2	2
Displace electrically		-	2	-	-	1	-	-	2	-
Recuperate electrically		1	-	-	1	-	-	1	-	-
Boost		-	-	1	1	-	2	3	-	1

The number of solutions for the four system boundaries defined using this method are shown in Table 4. The shown number of solutions is limited to so-called unique ones. Every problem element has a set of such fundamentally unique solutions, e.g. the one shown in Figure 48 for the corresponding high-level function "Boost". These unique solutions can be enhanced by certain structures (see Figure 46) to create derivative solutions. This is illustrated in Figure 47 for one unique solution of the high-level function "Boost".

Table 5: Result of both approaches, limited to unique solutions

	Number of unique solutions	
Function	this work	Helms and Shea [27]
Displace conv.	7	2
Displace elec.	5	2
Recuperate elec.	3	1
Boost	8	2

In Figure 48, the unique solution that was found for the scope of 11/30 is presented. On first sight, it seems redundant. However, assuming that the power-splitting planetary

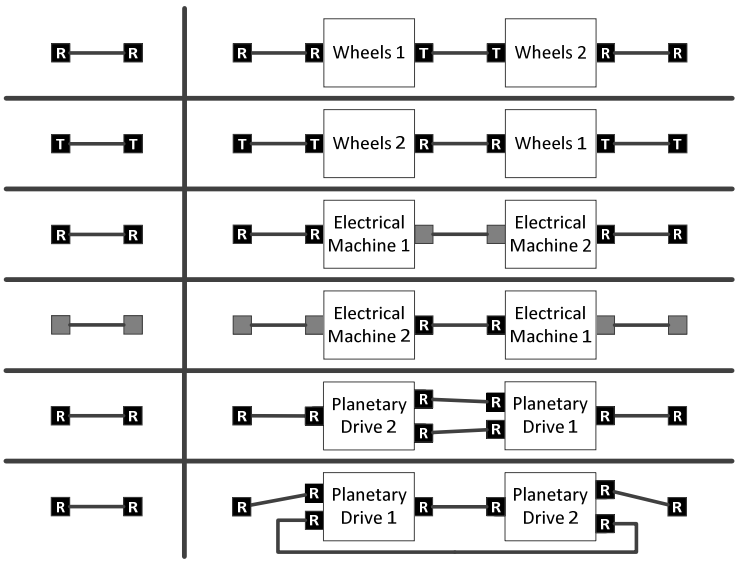


Figure 46: Structures enhancing the unique solutions. The subtopologies on the left-hand side can be replaced by those on the right-hand side.

drive can switch between both outputs using brakes, this solution defines a hybrid powertrain, offering four different transmission ratios for conventional driving, one for electric driving, and four for hybrid driving, using the combustion engine and electric engine at the same time. So, there are nine different paths through the CMG to get power to the wheels. These paths can be interpreted as speeds in a gearbox. Thus, including all planetary drives would create a 9-speed-gearbox for hybrid drives. Since such a device was not introduced until now and no research towards this is known to the author, this result shows the capability of this method to spark innovative solutions in engineering.

Table 5 shows this work's results on the left-hand side and those from the method of Helms and Shea [27] on the right-hand side. To enable a comparison, their solutions were also reduced to the unique ones. The comparison between the topological synthesis of this work and the approach of Helms and Shea [27] reveals the following: Helms and Shea's [27] approach is not able to generate all solutions that can be derived from the metamodel. This is not surprising given the simple search used, i.e. random walk. Disregarding the unique solutions, the approach only created a small portion of the possible designs. None of the highly intertwined solutions could be generated. This is due to the fact, that Helms and Shea [27] only build chains of elements with at most one branch point.

Another disadvantage of the approach of Helms and Shea [27] is using a random selection of elements to create chains. Due to this, the generation of all, or even one, solution can never be guaranteed, even if replaced by more sophisticated stochastic search. Although, using a high number of iterations, a particular degree of certainty can be reached. This approach, in contrast, uses the derived SAT problem to determine if the design task at hand is solvable for a specific scope. So, it offers a determination of the solvability of PMPs at specific scopes and with a defined metamodel. Further, if it is solvable, all solutions to the problem for a specific scope can be generated. Increasing the scope is only limited by the time needed for calculation.

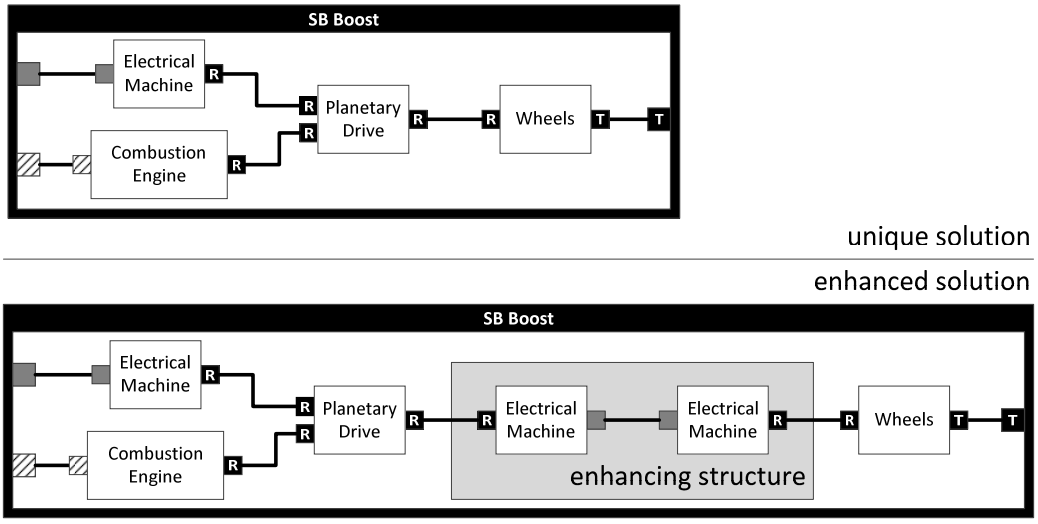


Figure 47: Comparison between a unique and an enhanced solution.

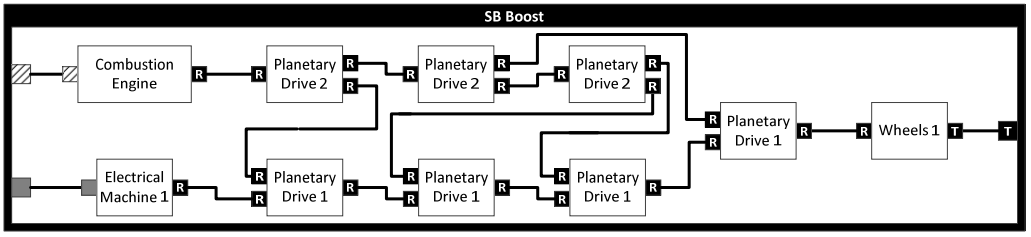


Figure 48: Unique solution (Scope: 11/30, calculation time: 28min).

The presented topological synthesis method offers a more systematic approach to computational design synthesis. Starting with the smallest calculated scope to create the smallest solutions first, and then going step-by-step towards more complex solutions by increasing the scope the solution space can be explored systematically. Further, this thesis' approach is able to generate more solutions due to its capability to generate highly intertwined topologies, or networks, and proves its capability to generate potentially innovative solutions. So, this approach is a logical next step coming from a purely rule-based method to rule- and model-based approaches to a purely model-based approach in combination with sophisticated logical reasoners.

However, the approach of Helms and Shea [27] offers the possibility to define problem-specific rules in addition to the generic rules that are used for the synthesis. These rules incorporate further engineering knowledge for the specific design task. So, disadvantageous solutions can be prohibited or changed into more advantageous ones. However, the concept modeling framework used in this thesis is based on the one presented in [27] and, theoretically, still supports the application of problem-specific rules. Due to the intentions, to release the designer from defining rules, this possibility is not considered in this work's approach, but including such rules could render advantages in specialized applications.

6.3 Case Study III - Automotive Powertrains II

This section presents another application of the presented method for the case study of automotive powertrains. But in contrast to Section 6.2, here also the generation of variable assignments and optimization using simulated annealing, thus the whole method presented in Section 4 is performed. Therefore, this case study gives a proof-of-concept for the method and its applicability also for the evaluation of technical systems of level of complexity III. The following sections will first describe the metamodel and then give details about the method application before the results are presented and selected powertrain concepts are discussed.

6.3.1 Metamodel

The port hierarchy used in this case study is shown on the left side of Figure 49. Apart from the energy ports, four different signal ports are defined:

- Acceleration signal port: transmits the acceleration pedal position between 0 and 1.
- Brake signal port: transmits the brake pedal position between 0 and 1.
- Gear signal port: transmits the gear that is to be enabled if a gearbox is present $\{1, 2, 3, 4, 5, 6\}$.
- Engine speed signal port: transmits the rotary speed at the input shaft of the gearbox.

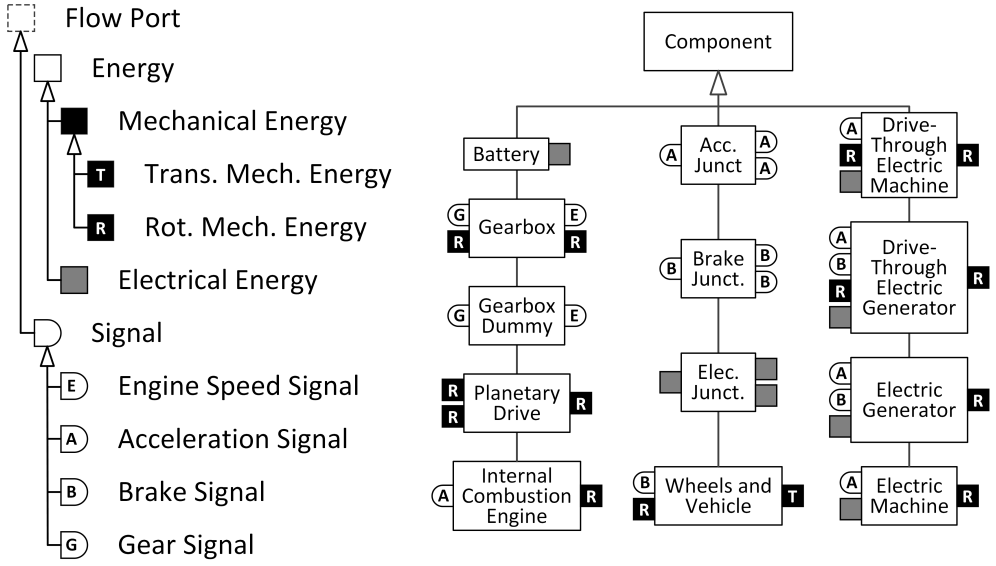


Figure 49: The port hierarchy on the left and the defined components on the right.

The metamodel is illustrated in Figure 49. Thirteen different components with parameters, constraints and partial simulation models are modeled as described in Section 4.1. These components are presented in the following paragraphs.

Gearbox This element models a six speed manual gearbox with included clutch. The gear to be engaged is transmitted at the signal in-port while the rotational speed at the input flange is transmitted at the signal out-port. For the parameterization and optimization, the component's variables are the transmission ratios of the six different forward speeds of the gearbox.

Gearbox Dummy This component is defined to allow powertrain concepts without gearboxes although gearbox-specific signals are required by the system boundary. The gearbox dummy component takes a gear signal at the in-port and returns a default engine speed signal at its out-port.

Planetary Drive The planetary drive component in this case study has two rotational mechanical energy in-ports representing the sun gear and the planet carrier and one rotational mechanical out-port that represents the ring gear. This component's variables are the ring and the sun gear radius.

Battery The battery element represents a rechargeable battery with one electric energy out-port. In the associated partial simulation model the total discharge E_{el} the battery undergoes during the drive cycle is recorded using Eq. 61 where P_{el} is the charging and discharging electric power.

$$E_{el} = \int P_{el} dt \tag{61}$$

Internal Combustion Engine This component represents a four-cylinder four-stroke gasoline internal combustion engine. It has an acceleration signal in-port where the pedal position is transmitted and a rotational mechanical energy out-port representing the engine's output flange. In its partial simulation model, the cumulated CO_2 emission is recorded.

Wheels and Vehicle This component represents the differential, two wheels, the chassis and body work of the car. The rotational mechanical energy that is transmitted at the in-port drives the rear axle after going through the differential. The transmission ratio of the differential is the component's variable. Additionally, the brake signal which is expected at the signal in-port controls the brakes of the car. The movement of the car which is generated at the footprint of the driven wheels is represented by the translational mechanical energy out-port.

Acceleration Junction (Acc. Junct.) This is a signal junction that copies the acceleration signal at the in-port to both out-ports.

Brake Junction (Brake Junct.) The brake junction is a signal junction that copies the brake signal at the in-port to both out-ports.

Electric Junction (Elec. Junct.) At this energy junction electric energy is distributed according to Kirchhoff's current law between the electric energy in-port and the two electric energy out-ports.

Drive-Through Electric Generator This component represents an electric engine that is able to both, a) increase a rotational mechanical energy flow by superimposing positive torque generated from electric energy and b) decrease a rotational mechanical energy flow by superimposing negative torque and generating electric energy. Therefore, the drive-through electric generator has both, one rotational mechanical in- and one out-port. The gas pedal position is transmitted at the acceleration signal in-port and electric energy at the electric energy in-port. Its variable is the nominal power of the embedded electric machine.

Drive-Through Electric Machine In contrast to the drive-through electric generator, this electric engine is only able to increase a rotational mechanical energy flow by superimposing positive torque generated from electric energy. It has an acceleration signal in-port, a electric energy in-port and each one rotational mechanical in- and out-port. Its variable is the nominal power of the embedded electric machine.

Electric Generator The electric generator component models an electric engine that is capable of both, transforming an electric energy flow into a rotational mechanical one, and vice versa. Electric energy is transmitted at the electric energy in-port while the engine's flange is represented by one rotational mechanical out-port. The gas pedal position is transferred at the acceleration signal in-port. The electric machine's nominal power is the component's variable.

Electric Machine In contrast to the electric generator, this element represents an electric engine that is only capable of transforming an electric energy flow into a rotational mechanical one. Apart from the electric energy in-port and the rotational mechanical out-port it also has an acceleration signal in-port and the machine's nominal power as variable.

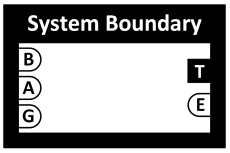


Figure 50: System boundary for hybrid powertrains

System Boundary The system boundary is shown in Figure 50. Basically, it represents a driver who wants to follow certain speed profiles, once the urban driving cycles (UDC) and once the extra-urban driving cycle (EUDC) from the New European Driving Cycle (NEDC) [109]. The NEDC is a defined speed profile which is used in the European Union to measure fuel consumption and emissions from cars in order to classify them in different energy efficiency classes. The speed profile used in this case study and the operation points used for the generation of variable assignments are shown in Figure 51.

To follow this speed profile, the system boundary provides an acceleration signal, a brake signal, and a gear signal. For the concept a translational mechanical energy flow is required and an engine speed signals to trigger the shifting of gears in a possible gearbox. The objective function for the simulated annealing optimization is defined to minimize the cumulated error e_i between the actual driven speed (v_{actual}) and the required one (v_{demand}) over the driving cycle (Eq. 62). Therefore, the optimization model shown in Eq. 63 is defined in the system boundary, with *x* being a variable assignment for the generated solution, V being the set of variable assignments generated for the solution, $\mathit{size}(V)$ defining the size of V, i.e. the number of variable assignments to be created, and k being the amplifier for simulated annealing..

$$e_i = \int |v_{demand} - v_{actual}|\, dt \tag{62}$$

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & e_i(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in V, \\ & \mathit{size}(V) = 10000, \\ & k = 1 \end{aligned} \tag{63}$$

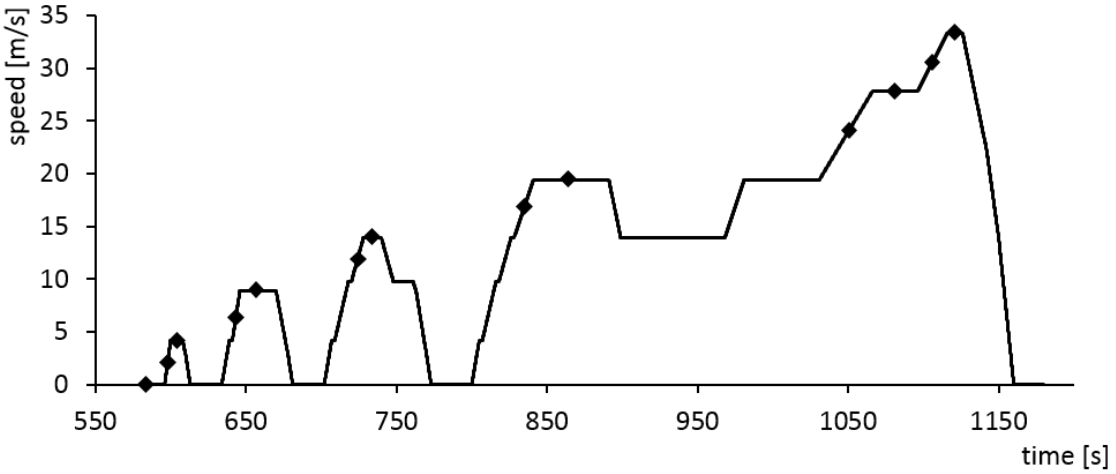


Figure 51: The UDC ($580s \leq t < 780s$ in the NEDC) and the EUDC ($780s \leq t < 1180s$ in the NEDC). The operation points used in the case study are indicated with black diamonds.

6.3.2 Method Application

For the topological synthesis, the maximum number of components within the powertrain is set to ten. Following the calculations presented in Section 4.2.2, seven different component categories are identified and 1189 different scopes result. For these scopes, the first-order logics and following that the Boolean satisfiability problems are generated and, consecutively, solved using the high performance cluster of ETH Zurich with the restriction of a maximum calculation time of four hours per scope. This time showed to enable moderate waiting times on the server with sufficient solution space exploration. Since not all scopes required the maximum time, the total CPU time used for the topological

synthesis was approximately 2500 hours.

In the end, 162 different solutions containing only CMGs are generated. Following the method, the CSP for each solution is generated and solved, generating 10000 variable assignments for every topology where possible. For 43 topologies, no valid parameterization exists. Now, all the 119 remaining topologies are optimized to best meet the defined speed profile. For 28 topologies, even though the CSP could be solved, no variable assignment could be found, that could be simulated without error. All these 28 powertrain concepts include at least one internal combustion engine and a planetary drive. So, although, all operation points defined in the system boundary can be realized in the powertrain according to the variable assignment generation, due to dynamic effects in the interplay between the internal combustion engine and another source of torque over the planetary drive in the simulation model, the internal combustion engine is forced to an operational state where it would have to run backwards. This causes the simulation to abort, since the model of the internal combustion engine cannot handle backwards rotation. In the end 91 powertrains were successfully optimized setting the amplifier k (cf. Section 4.4) to one for the change of the objective function and 500 iterations. On average the simulated annealing process converges after approximately 200 iterations. The total CPU time for the optimization of all 91 powertrains was approximately 400 hours.

6.3.3 Results

To compare the different optimized concepts, the following performance values are used: first, the total CO_2 emissions generated during the driving cycle. For conventional powertrain concepts this is the cumulated CO_2 emissions of the combustion engine(s). For electric powertrain concepts the electric energy that was discharged from the battery(s) is multiplied by 522 g CO_2/kWh [110], to take into account the CO_2 emissions during the electricity production. Hybrid powertrain concept emissions are calculated using both calculations. The second performance value is the objective function, the cumulated error from the speed profile.

The solution space is shown in Figure 52. As can be seen, using more than one internal combustion engine significantly increases the CO_2 emissions. Excluding the 27 solutions containing two or more internal combustion engines results in a reduced solution space. It is shown in Figure 53. Here, the dominant sets regarding cumulated error and total CO_2 emission for each powertrain type (conventional, hybrid, electric) are illustrated in black.

In Figure 53 four example powertrain concepts (indicated with (1)-(4) in Figure 53) are indicated which are described in more detail and discussed in the following paragraphs. The results for the four concepts are comprised in Table 6 and the convergence of the objective function value averaged over 4 runs is shown in Figure 54.

Solution 1 (see Figure 55) represents a conventional powertrain. An internal combustion engined is connected to a gearbox which is itself connected to the differential that drives the wheels within the wheels and vehicle component. The cumulated error from the

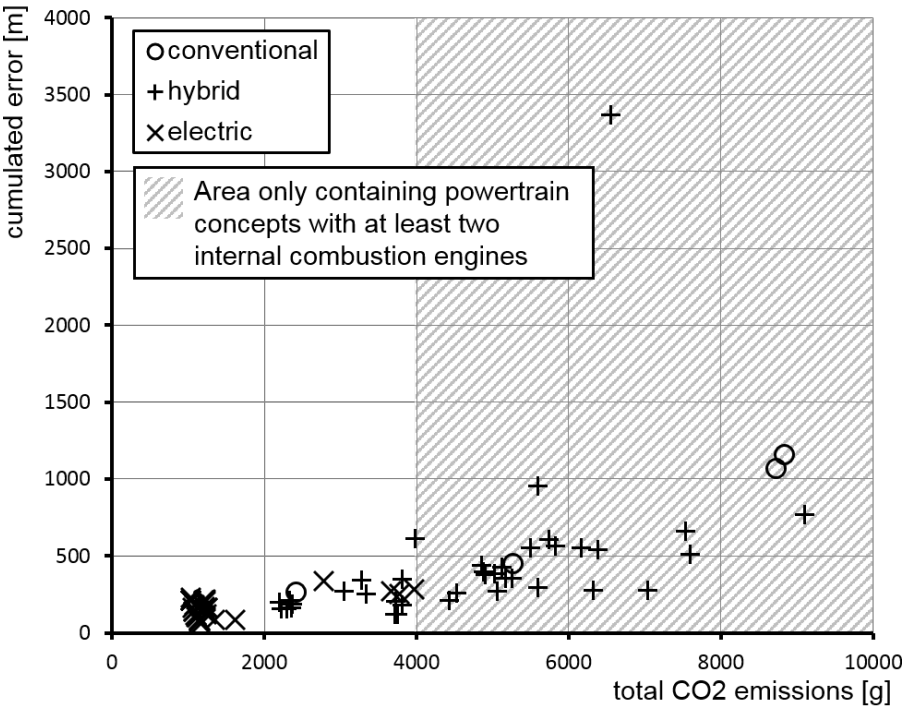


Figure 52: Complete solution space. The hashed region marks the area in which only powertrain concepts are found that utilize at least two internal combustion engines.

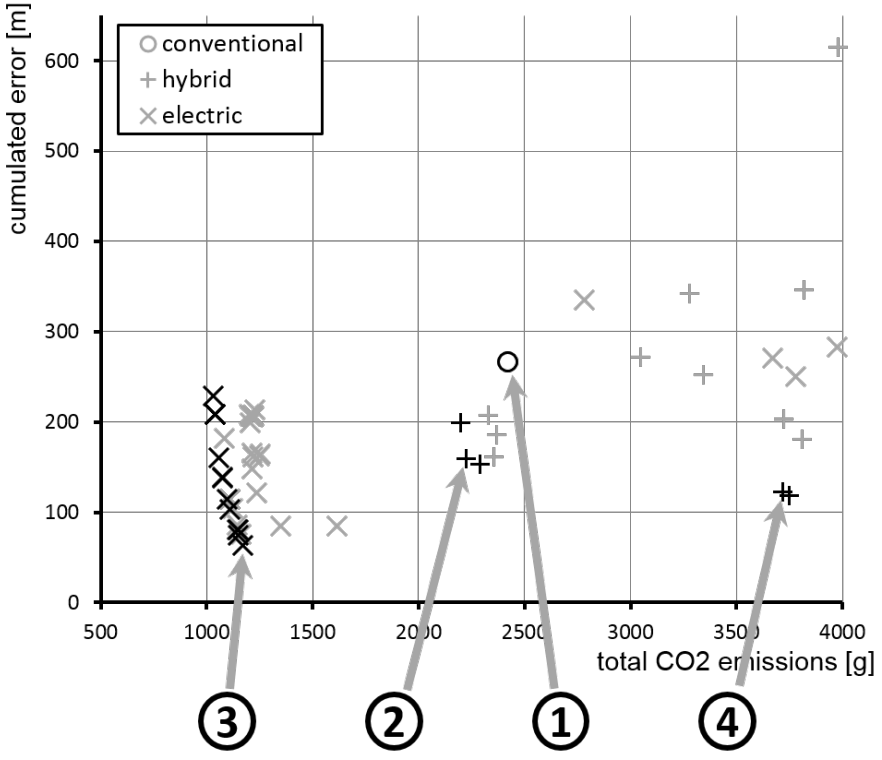


Figure 53: Solution space excluding concepts with two internal combustion engines. The dominant concepts with regard to cumulated error and CO_2 emission are indicated in black for each powertrain type, the dominated ones in grey.

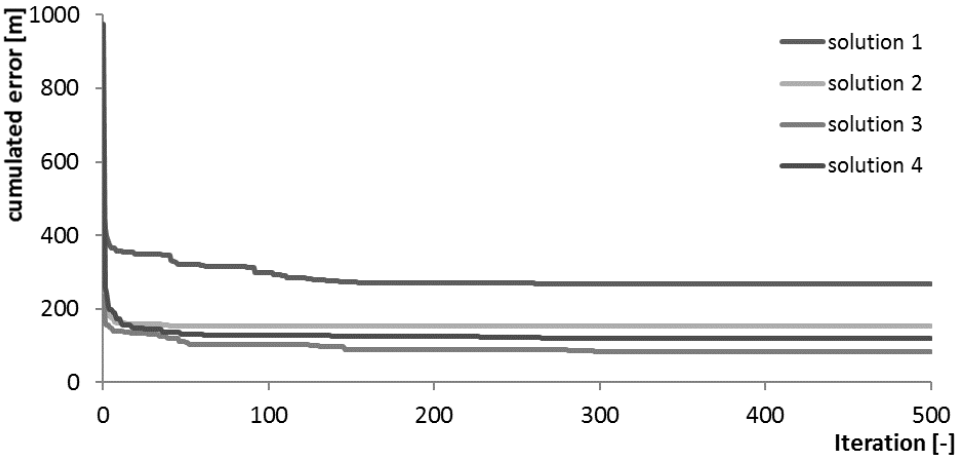


Figure 54: The convergence of the objective function of solution 1-4 averaged over 4 runs with each 500 iterations.

Table 6: Comprised results for the example powertrain concepts 1 – 4.

Solution	1	2	3	4
	Figure 55	Figure 56	Figure 57	Figure 58
Cumulated error	267 m	154 m	63 m	118 m
Total CO_2 emissions	2419 g	2290 g	1171 g	3751 g

given speed profile is approx 267 m while emitting a total of approximately 2419 g CO_2. LMS Amesim provides a reference model for a conventional automotive powertrain. This model was validated on a real automotive powertrain. Comparing the generated solution to this reference model reveals the following. Simulating the reference model using the same speed profile results in a cumulated error of approximately 69 m and a total CO_2 emission of approximately 2154 g. The better values of the reference model are due to all the components in the reference model being carefully matched to each other. That means, that not only the design variables that are optimized in the metamodel are adapted to this specific use case, but also further variables are assigned to support that specific powertrain concept. Therefore, the reference model represents a final design in contrast to a conceptual design (cf. Section 1). Taking this into account, it can be concluded that the presented method can give a good indication of the performance of the different concepts.

Figure 56 illustrates solution 2. Here, the internal combustion engine is supported by two electric engines, a drive-through electric machine in front of the gearbox, and a drive-through electric generator behind the gearbox. Simulating this hybrid powertrain concept results in a cumulated error of approximately 154 m and a total CO_2 emission of approximately 2290 g. Compared to the conventional solution (1), the two electric machines help to achieve a lower cumulated error. Further, the support of the internal combustion engine in disadvantageous operation states and the rudimentary possibility to recuperate energy of the electric generator help obtain a lower total CO_2 emission even though the used electric energy is penalized with 522 g CO_2/kWh [110].

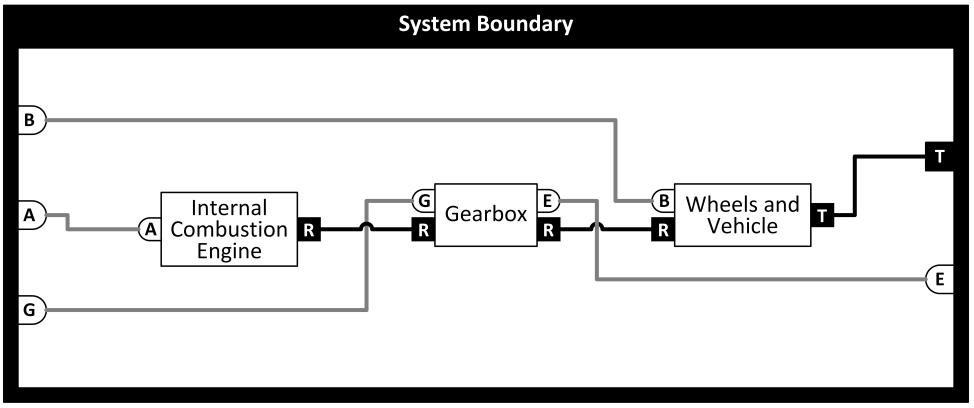


Figure 55: Solution 1 - conventional powertrain - total CO_2-emissions: 2419 g - cumulated error: 267 m (only for illustration purposes, the relations connecting signal ports are colored grey and those connecting energy ports are colored black)

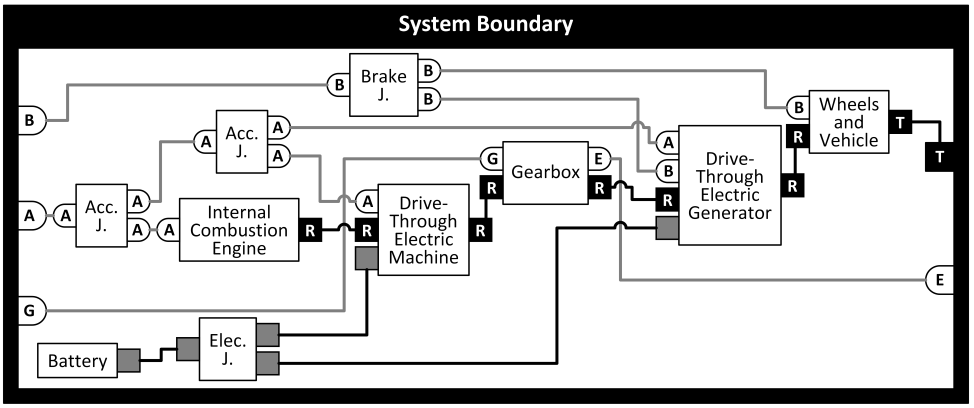


Figure 56: Solution 2 - hybrid powertrain - total CO_2-emissions: 2290 g - cumulated error: 154 m (only for illustration purposes, the relations connecting signal ports are colored grey and those connecting energy ports are colored black)

The pure electric concept that generates the smallest cumulated error of approx. 63 m is shown in Figure 57. Basically, this powertrain resembles solution 2 without the internal combustion engine and the recuperation capabilities, since another drive-through electric machine is used instead of the drive-through electric generator that is placed between the gearbox and the wheels and vehicle component. For such a small error, the electric machine is favorable over the generator. This is due to the recuperation's negative torque in combination with the conventional brakes resulting in a deceleration that is steeper than the intended one. Therefore, the total CO_2 emission could be reduced by using a generator instead of the drive-through electric machine in combination with a better control strategy.

The hybrid automotive powertrain represented in solution (4) is shown in Figure 58. Here, the gearbox dummy component is used to connect the given input to the output of the system boundary only, thus providing a complete solution according to the SAT problem even though no gearbox is present. In this CMG, the internal combustion engine is supported by one electric machine and a drive-through electric machine. The internal combustion engine and the electric machine are placed on one side of a planetary drive whereas the drive-through electric machine is on the other side in front of the wheels

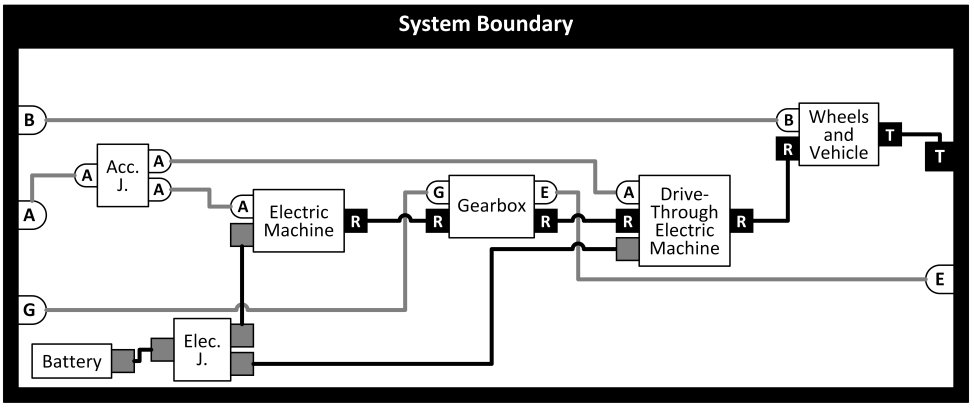


Figure 57: Solution 3 - electric powertrain - total CO_2-emissions: 1171 g - cumulated error: 63 m (only for illustration purposes, the relations connecting signal ports are colored grey and those connecting energy ports are colored black)

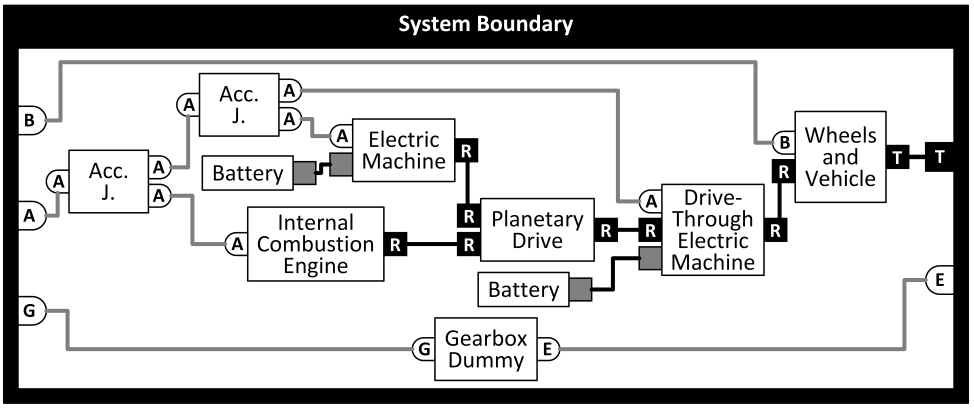


Figure 58: Solution 4 - hybrid powertrain - total CO_2-emissions: 3751 g - cumulated error: 118 m (only for illustration purposes, the relations connecting signal ports are colored grey and those connecting energy ports are colored black)

and vehicle component. Therefore, solution (4) resembles the powertrain that is implemented in the Toyota Prius II. For this vehicle Toyota advertised a CO_2 emission of 104 g CO_2/km [111]. From the results from the simulation of solution (4) a CO_2 emission of 349 g CO_2/km is calculated. For this calculation only the emissions from the internal combustion engine are considered, since the official measuring procedure only accounts for local CO_2 emission. Local CO_2 emission are emissions that are produced directly in the vehicle. The emissions from the generation of the electric energy are not considered. This shows, that the results from the hybrid powertrains can only be used to a certain extent to compare the different powertrain concepts since no real hybrid driving modes are considered and controlled. When the driver in the system boundary wants to accelerate, due to the absence of an adapted control allocation strategy, the acceleration signal is simply forwarded to all possible receiving components and not only to the most opportune ones to reduce CO_2 emissions. Since the Toyota Prius only produces less than 30 % of the CO_2 emission of solution 4, the potential of "intelligent" control strategies can be seen. Therefore, future work should consider generic ways to optimize the control allocation and, by that, the CO_2 emission independent from the powertrain layout to which it is applied.

6.4 Case Study IV - Printhead Drive for 3D Printers

This case study deals with the conceptual design and optimization of kinematics and drives for printheads for 3D printers. It is based an undergraduate student group project. Its task was to develop a 3D printer that could print faster than 3D printers that were available at that time. In order to choose an appropriate kinematic mechanism for the printer, different possibilities of kinematic mechanisms had to be evaluated. In the project, this was achieved by comparing the average kinetic energy in the kinematic mechanism while moving the printhead along a predefined profile [112]. This thesis' method investigates different concepts to achieve as high as possible print speed with a given precision. This case study is conducted to show the possible application of the method in a design project, the generation and evaluation of technical systems of complexity level II, and to show an application on a two-dimensional mechanism. In the following sections, first, the metamodel will be presented before the method application and the results are shown and discussed.

6.4.1 Metamodel

The metamodel for this case study consists of eight different components that can be used in the topological synthesis: two kinematics (two-slider kinematic and scissors kinematic) and six different drives as can be seen in Figure 59. In order to enable the possibility of a topological synthesis using the method presented in Section 4.2, the different drives have to be defined separately for two different connection points to the kinematic mechanism. This is done, since the two kinematic mechanisms each need drives at two different connection points and it has to be assured that the control signals that are calculated for the drives of each connection point actually reach the drive that is connected to it. Consecutively, this differentiation must also be reflected in the port taxonomy that is shown in the left part of Figure 59. The different elements are described in the following sections.

Kinematic In this case study, the kinematic mechanism moves the printhead of the 3D printer. A kinematic mechanism has two connection points (translational mechanical in-ports) for the drives that actuate the movement of the printhead. The control signals (Machine Signal 1 and 2 ports) for these drives are calculated from the specified x- and y-position of the printhead. The actual position of the printhead is transmitted at the x- and y-position signal ports. The two kinematic mechanisms available for the CMGs are presented in the following paragraphs.

Two-Slider Kinematic The first kinematic mechanism is a two-slider kinematic as illustrated in the left part of Figure 60. One slider is moved to achieve movement in y-direction and a second, smaller slider is driven on the bigger slider for movement in x-direction. Here, the demanded x- and y-position are also the machine signals one (pos_1) and two (pos_2): $pos_1 = x_{dem}$ and $pos_2 = y_{dem}$. The partial simulation model of the two-slider kinematic is shown in Figure 61. At the bottom, the specified x- and y-positions are converted into the demanded machine positions using a gain of one. The main part of the partial simulation model consists of two bodies that are constrained to move along

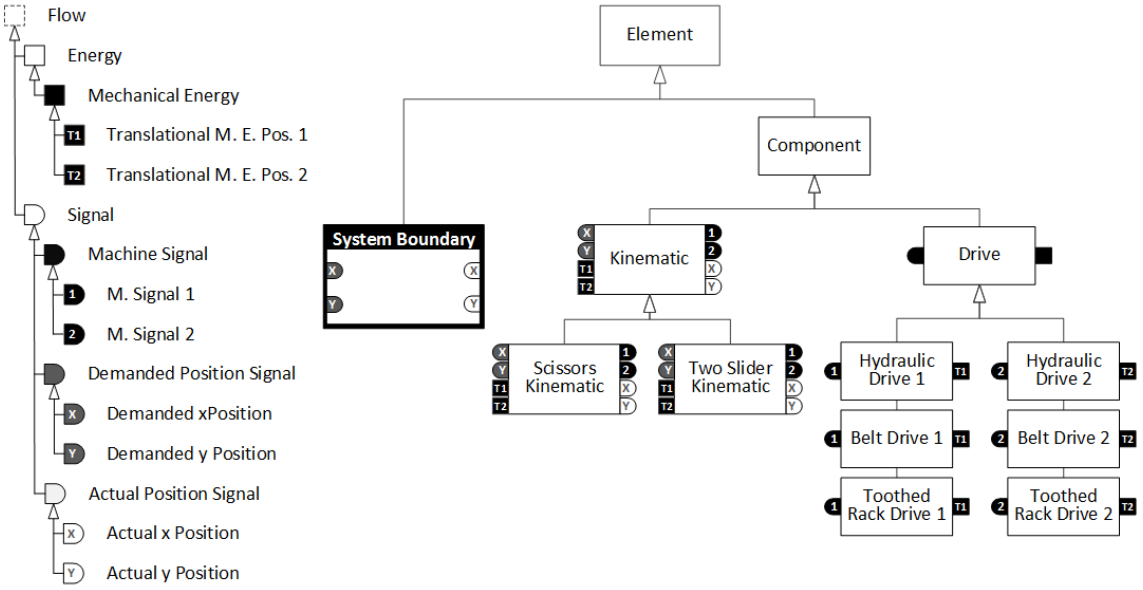


Figure 59: The metamodel of available elements and the port hierarchy for the printhead drive case study.

the y-axis and the x-axis of the first body respectively. The second body represents the printhead. Two sensors are used to obtain the x- and y-position and two other sensors for the x- and y-speed. The two speeds are used to calculate the absolute speed of the printhead $v_{printhead}$ (Eq. 64). $v_{printhead}$ is recorded during the simulation.

$$v_{printhead} = \left| \sqrt{v_x^2 + v_y^2} \right| \tag{64}$$

Scissors Kinematic The scissors kinematic shown in the right part of Figure 60 is the second kinematic mechanism used in this case study. Here, the printhead is located on the hinge joint between the two arms. The other ends of these arms can be moved in the x-direction in order to allow a free movement of the printhead. As in the previous kinematic mechanism, the needed machine positions are derived from the demanded x- and y-position of the printhead. The machine positions are calculated using Eqs. 65 and 66, where a is the length of the arms and ω_0 their initial angle to the x-axis (cf. Figure 60). In this kinematic mechanism the absolute speed of the printhead is calculated from its x- and y-speed (cf. Eq. 64) and recorded during the synthesis.

$$pos_1 = (x_{dem} + a\cos(\omega_0)) - \left| \sqrt{a^2 - (y_{dem} + a\sin(\omega_0))^2} \right| \tag{65}$$

$$pos_2 = (x_{dem} - a\cos(\omega_0)) + \left| \sqrt{a^2 - (y_{dem} + a\sin(\omega_0))^2} \right| \tag{66}$$

Drive The drive components represent different ways to actuate the movement of the connection points to the kinematic mechanism. Therefore, each drive has a machine signal input, that actually describes the needed displacement at the connection point that is represented by a translational mechanical energy port. In this case study, each drive is

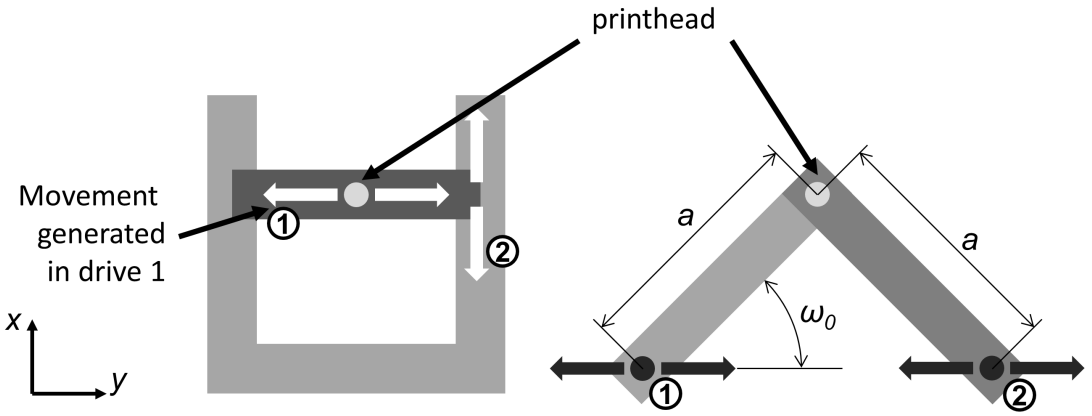


Figure 60: Schematic drawings of the two basic kinematic mechanisms: the two-slider kinematic on the left and the scissors kinematic on the right. The numbers 1 and 2 indicate the movement that is generated by the drive at connection point one or two, respectively.

internally powered by electric energy and the total amount of electric energy E_{el} that is used during a simulation is calculated using Eq. 67 where P_{el} is the measured electric power. To allow the evaluation of the demanded energy in the simulation E_{el} is recorded throughout the simulation.

$$E_{el} = \int |P_{el}|\, dt \tag{67}$$

Although six different drive components are defined in the metamodel, only three distinct types exist. The two versions of each drive type (1 and 2) only differ in the instantiated port types but the constraints and partial simulation models are the same. This is done, to prevent solutions where a machine in position two receives machine signals intended for position one. Each type is only presented once.

Hydraulic Drive The hydraulic drive models an electric engine that drives a pump. This controllable pump drives a hydraulic cylinder that realizes a translational displacement that is to be transferred at the output of the system. The design variables are the expected rotary speed of the pump n_{dem} and its displacement per revolution V_{dis}. Additionally, the expected rotary speed n_{dem} affects the electric engine that has to provide this speed. The product of n_{dem} and V_{dis}, i.e. the pump's flow rate, has to be higher than $2 \times 10^8 \frac{m^3}{s}$ to assure the unit's operation and avoid unwanted vibrations. This value was experimentally determined during the design of the drive type.

Belt Drive The belt drive is a controlled electric engine that is connected to a pulley. Between that driven pulley and a non-driven pulley a belt is taut. On this belt a mass that models the connection point of this drive, is pulled or pushed to the position given by the input signal. The element's variable is the radius of the driven pulley.

Toothed Rack Drive The toothed rack drive shares the underlying setup of the belt drive. The rotational motion from the electric engine is transformed into translational

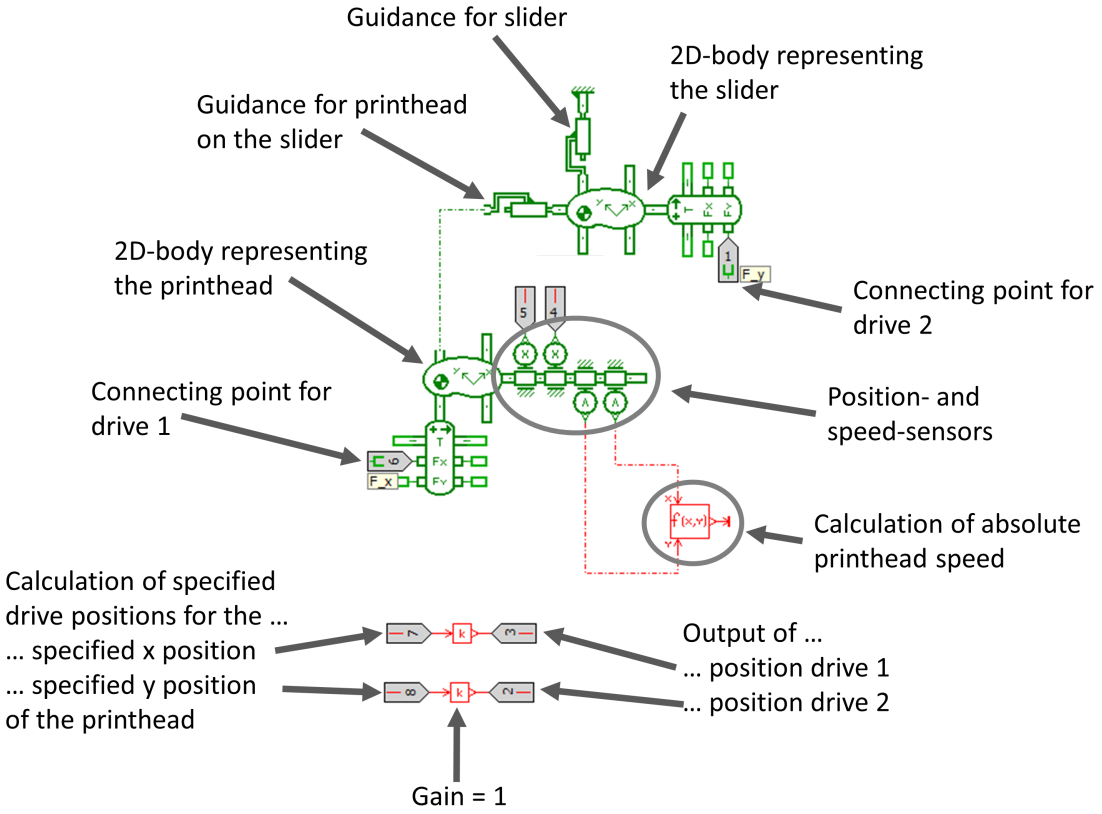


Figure 61: Partial simulation model of the two-slider kinematic in LMS Amesim.

motion by a pinion gear that drives a toothed rack. The variable is the radius of the pinion gear.

System Boundary As described in Section 4, the system boundary describes the environment of the concept. The outputs to the concept are the specified x- and y-positions of the printhead and the inputs of the system boundary are the measured x- and y-positions of the printhead.

At the beginning of the simulation, the printhead is in the center position ($x = 0$ and $y = 0$) and, starting there, is to drive a circle with a radius of 0.1 m and a constant frequency of 0.2 Hz. After 5 seconds, i.e. one full circle, the speed is linearly increased until the simulation stops. The stopping criterion is a deviation between the demanded and actual position of the printhead that is more than 1 mm. This deviation Δ is calculated using Eq. 68 where x_{act} and y_{act} are the actual and x_{dem} and y_{dem} the demanded x- and y-positions of the printhead.

The objective for the optimization is to maximize the speed of the printhead v (Eq. 69) with a maximum deviation of the given printhead path of 1 mm. Since the method is designed to minimize an objective, the sign is reversed for the always positive speed v. The optimization model shown in Eq. 70 results, with x being a variable assignment for the generated solution, V being the set of variable assignments generated for the solution,

and $size(V)$ defining the size of V, i.e. the number of variable assignments to be created.

$$\Delta = \left| \sqrt{(x_{act} - x_{dem})^2 + (y_{act} - y_{dem})^2} \right| \tag{68}$$

$$v = \left| \sqrt{\dot{x}^2 + \dot{y}^2} \right| \tag{69}$$

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & -v(\mathbf{x}) \\ \text{subject to} \quad & \mathbf{x} \in V, \\ & size(V) = 20000, \\ & k = 10000 \end{aligned} \tag{70}$$

6.4.2 Method Application

Performing the topological synthesis as described in Section 4.2 results in a total of 18 different concepts, the maximum number of combinations for two basic kinematic mechanisms with each two connection points and three different types of drives ($2 \cdot 2 \cdot 3 = 18$). For parameterization and optimization, six concepts are investigated. These concepts are each powered by only one drive type (hydraulic, belt, or toothed rack). Mixed concepts are not considered since the use of two different types in one printhead drive is not desired.

The parameters for the simulated annealing process are set as follows. The amplifier k for the change of the objective function value between the iterations is set to 10000 (cf. Section 4.4). This is due to the output of the objective function being calculated in $\frac{m}{s}$ but only small changes in the order of magnitude of $\frac{mm}{s}$ are expected between the different iterations. Given the increase of 0.125 $\frac{m}{s^2}$ and a time-step of 0.025 s, the expected difference between two data points is 0.003 $\frac{m}{s}$. Without the amplifier, small differences between the objective function values would push the acceptance function close to one (cf. Section 4.4). This would mean that almost all investigated variable assignments are accepted during the simulated annealing process. The size of the calculated set of variable assignments is 20000 and the number of iterations is set to 500.

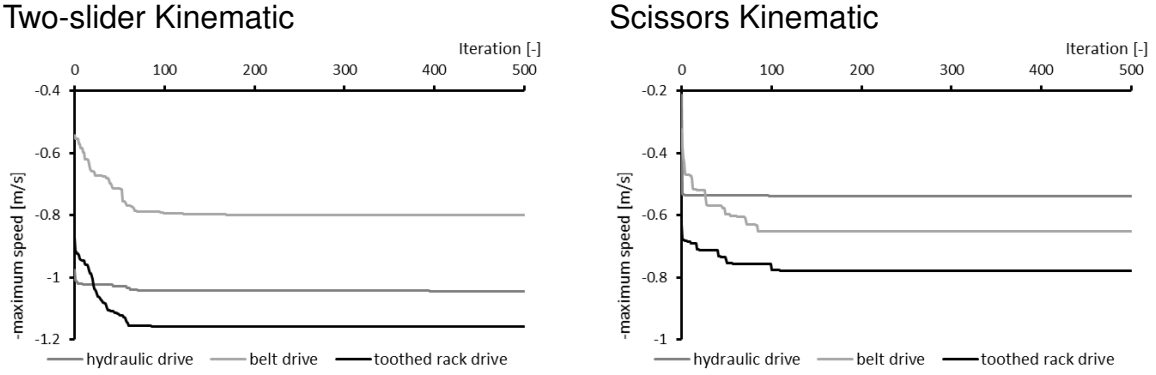


Figure 62: The development of the objective function value averaged over five simulated annealing runs with different starting points.

To assure convergence and since simulated annealing is a stochastic method, the optimization process is performed five times for each of the six solutions using different initial variable assignments. The development of the different runs for each solution are illustrated in Figure 62. It can be seen, that the concepts converge reliably within the 500 iterations.

6.4.3 Results

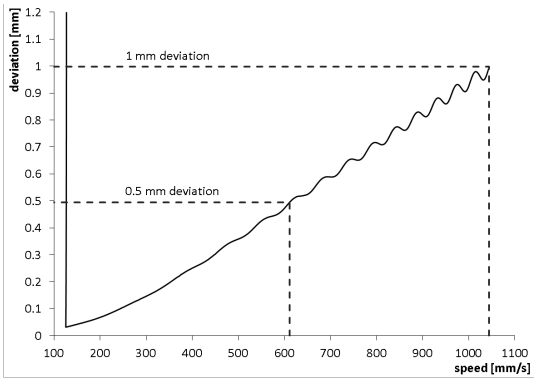
In Figure 63, the printhead's deviation Δ from the intended path (cf. Eq. 68) over the printhead speed is shown for all investigated concepts. The deviation for the concepts using the scissors kinematic rise more nonlinearly than the deviation of concepts using the two-slider kinematic. This is due to the superposition of rotational and translational accelerations in the scissors kinematic which leads to a more complex dynamic behavior than the strictly translational motions in the two-slider kinematic. The oscillation of the two-slider concepts, especially becoming apparent at higher speeds, results from the control in the drives, which is not adapted during the optimization.

In Figure 64, the effect of the direction of motion on the deviation Δ is investigated. The graph shown for both kinematic mechanisms driven by a toothed rack drive is representative for all the other drives. The error of the two-slider kinematic is independent from the sense of direction of the circular path and, therefore, of the printhead's direction of motion. However, the scissors kinematic shows a different behavior when the sense of direction is changed. This means, that the deviation Δ of the printer is not only dependent from the speed of the printhead but also the direction of motion. This behavior is a disadvantage of the scissors kinematic since the effort to assure a certain precision in printing is higher than for the two-slider kinematic. The investigation, however, also shows that more than one load case should be simulated to enable a higher solid foundation for the decision towards one or the other concept. For this case study, not only circular paths, but also quadrangular paths at different positions of the printing table. Since the main intent of this case study was to show the applicability of the presented method, these use cases were not considered.

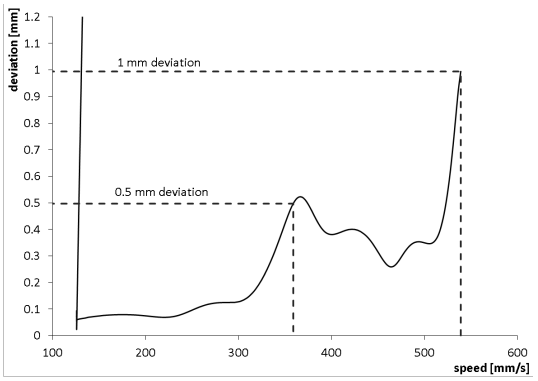
The different recorded values that result from the simulated annealing process (cf. Section 4.4) are shown in Table 7. E_{total} refers to the total energy consumption of the particular concept that was calculated by adding the used electric energy from every drive in the particular solution. $v_{max,0.5mm}$ and $v_{max,1mm}$ are the maximum speed of the printhead with a maximum deviation of 1 and 0.5 mm, respectively. As described before, $v_{max,1mm}$ was maximized during the simulated annealing process. $v_{max,0.5mm}$ was obtained by the simulation results as shown in Figure 63.

The results in Table 7 show that the two-slider kinematic with two toothed rack drives dominates all other solutions in every performance value. Further, the maximum speed is higher for the two-slider kinematic than for the scissors kinematic for all drive variants. Therefore, it can be concluded that a two-slider kinematic is better suited for investigation for fast, high-precision 3D printing. It must be noted that the control of the different drives

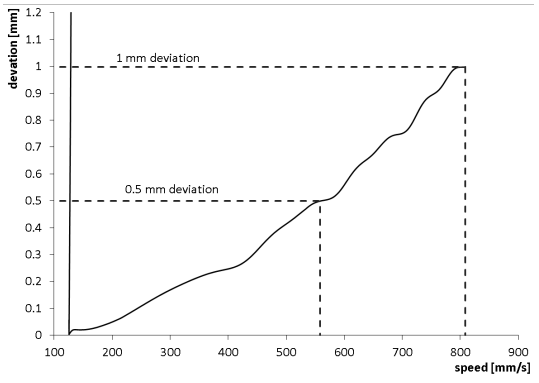
Two-slider Kinematic with hydraulic drive



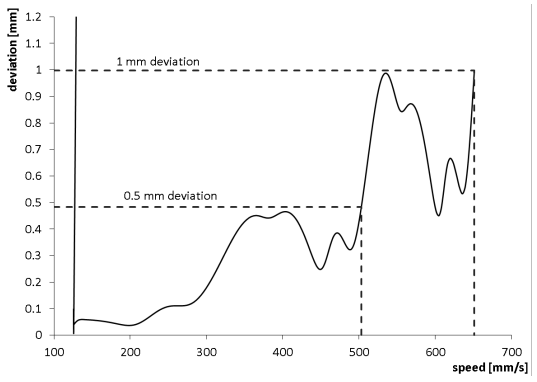
Scissors Kinematic with hydraulic drive



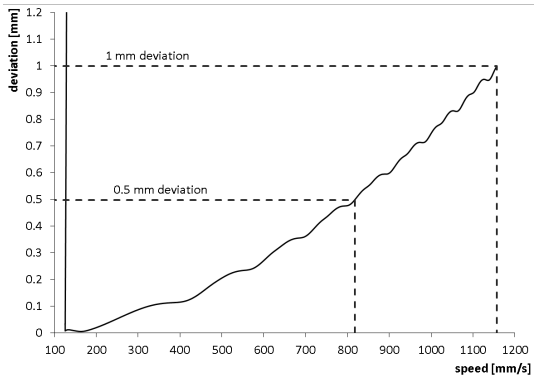
belt drive



belt drive



toothed rack drive



toothed rack drive

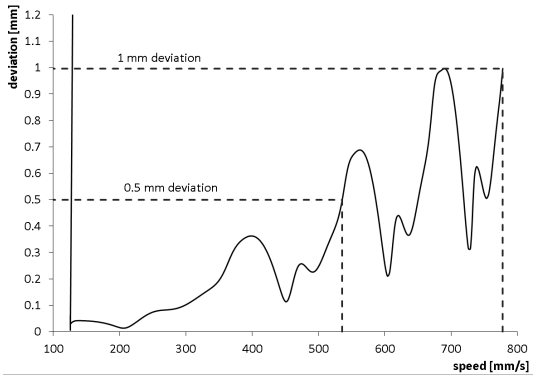


Figure 63: Diagrams showing the error between the printhead's intended and actual path over printhead speed. The points where $v_{max,0.5mm}$ and $v_{max,1mm}$ are obtained are indicated.

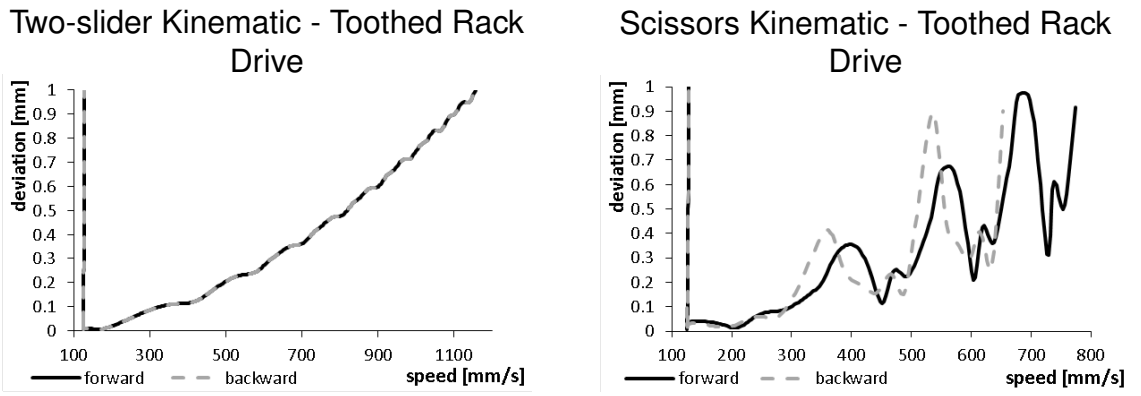


Figure 64: Diagrams each comparing the printhead's deviation for driving the circular path in opposite directions.

Table 7: Results for the investigated 3D printer kinematic concepts. The best results for each criterion is highlighted in grey.

	e_{total} [J]	$v_{max,1mm}$ [mm/s]	$v_{max,0.5mm}$ [mm/s]
Two-slider kinematic with			
hydraulic drive	13489.0	1044.27	611.20
belt drive	1611.0	808.95	558.67
toothed rack drive	975.8	1157.26	817.60
Scissors kinematic with			
hydraulic drive	1009.6	538.88	358.75
belt drive	1742.6	651.33	503.12
toothed rack drive	1158.3	778.08	535.88

was not changed during the optimization process. Optimizing the control variables could open another possibility to increase the printhead speed. However, the printing speed of a 3D printer is not only limited by the capabilities of the chosen kinematic mechanism. Melting and delivering the filament are also important factors that limit the speed in 3D printing processes.

In the student project, the two-slider kinematic was chosen for the final design. But, in contrast to the presented results, a belt drive was chosen. This was chosen since belt drive can be designed in a way that both electric engines that drive the belt are placed outside the moving parts. In the end, the project's final printer was able to print with a speed of 400 mm/s.

The method that is presented in this thesis could have supported the undergraduate student team by releasing them from manually modeling the different kinematic mechanisms and giving them simulation results. Those could have helped to chose an opportune kinematic mechanism and drive to solve the conflict between speed and precision instead of judging the different concepts using the average kinetic energy.

6.5 Summary

This section presents four case studies that show different applications for the method that is presented in this work. The two preliminary case studies (Section 6.1 and 6.2) are restricted to the topological synthesis, whereas case studies two and three use the whole method. The results show that energy- and signal-based technical system with levels of complexity II and III can be automatically generated and evaluated. Technical system with levels of complexity IV can be tackled using the topological synthesis only. Such systems, i.e. plants, are mostly not limited to energy- and signal-flow but also incorporate material-flow and, especially, spatial layout. Both can, in the current state, not be fully included in the approach. Conceptually, as shown in Section 6.1 for material flow, they can be considered but an evaluation, or even optimization, is currently not feasible. Another issue that affects the simulation is the issue of controlling the simulation models. In the studies presented in Section 6.3 and 6.4, only simple control strategies are used and not adapted to the different concepts. As mentioned in the respective sections, that leads to sub-optimal performance. The challenge in adapting control with this method is that the variable assignments are generated based on static equations. To effectively design a controller, the dynamic of the system has to be known which is unknown during the generation of the variable assignments. Therefore, a specific controller for every variable assignment would have to be designed based on the dynamics of the simulation model that is not included in the method. Either possibilities to effectively include this or generic strategies in optimal control theory [113] and control allocation [114] should be investigated in future work to allow a better evaluation of the performance of the generated concepts.

7 Discussion

The method presented in this thesis provides a continuous generic approach to generation, evaluation and optimization of concepts. A graph-based object-oriented modeling framework and bond-graph based simulation models are used to automatically generate, evaluate and, by that, explore solution spaces for engineering tasks in the domain of energy- and signal-based systems. Thus, it provides computational support in the conceptual design phase of the product development process. Further, this work responds to the demand for the utilization of different representations [18, 24] and automated translations between them [24]. The design tasks this method can handle have the following characteristics. The elements from which the concepts are created and their interfaces have to be defined and the design synthesis task must be formalized as a transformation from none to a finite number of inputs to none to a finite number of outputs as long as there is at least one in- or output. The behavior of the different components must be known and modeled as partial simulation models according to the guidelines developed in this thesis. Defining the input-output-relations for every component and the operation points for the system boundary is optional, though helps to generate promising variable assignments. This reduces the needed iterations in the optimization (more details on this are presented in Section 7.2).

The topological synthesis introduced in this thesis resolves disadvantages of the rule-based representations through a model-based representation offering a high adaptability and expressiveness. By introducing more sophisticated reasoning, i.e. conflict analysis and clause learning [84, 33], that is available for Boolean satisfiability problems, the drawbacks of the simple search methods within the model-based approaches are compensated. Combining a graph-based object-oriented concept modeling framework with logics also exempts the human designer from the manual definition of abstract logical statements [58, 57, 55] and from the definition of rules but enables the use of logical reasoning to synthesize product topologies. Further, this approach overcomes the limited scope of application that restricts the approach of Hendricks et al. [58, 59].

Apart from the differences to the approach of Helms and Shea [27] (cf. Section 6.2), the method presented in Section 4 differs from the approach of Wyatt et al. [47], by utilizing no constructs that cannot be translated to logics. In addition, the expensive manual definition of a design schema is omitted here. It is replaced by a more intuitive metamodel definition. However, whether this leads to a reduction in modeling time has to be examined. Component number constraints, direct connection constraints and fan out constraints can be transferred by the metamodel definition. Indirect connection constraints can only partly be transferred because, using this work's method, the number of paths between two ports cannot be defined in advance. A further advantage of this work is that no initial model has to be provided to start the generation of alternatives.

This work relates to C-K theory [56] by utilizing the expansion of knowledge space K to deduce all possible element combinations from the metamodel and the disjunction from K to concept space C to generate concepts for the given system boundary. Future work should investigate the advantages that could be achieved by also including the expansion

of C and the conjunction of C to K in CDS approaches. Engineering design tasks that are formulated as in the work of Braha [55] can also be tackled with this method. However, the approach of Braha [55] is, due to its written description, easy to comprehend for human designers. This is especially an advantage considering the inclusion of requirements.

Adding the evaluation by simulation enriches the expressiveness of the results, since this work is able to generate and simulate alternative concept topologies. This step is not undertaken by any of the literature mentioned so far [58, 57, 55, 58, 59, 27, 47]. The presented method overcomes the limitations in [74, 75, 76, 39] where the optimization is carried out by only changing parameters in predefined simulation models. However, [74, 75, 76] can make use of advanced control strategies since they rely on a fixed topology. These control strategies enable a more realistic evaluation of the generated powertrain concepts. Additionally, where Bayrak et al. [78] only change possible connections in a predefined framework of one combustion engine, two electric machines and two planetary drives, this work's method is capable of combining the available components to entirely different powertrain topologies detached from the fixed composition of components. However, Bayrak et al. are investigating possible driving modes to identify a beneficial hybrid control strategy in the powertrain. Their work can be considered a possible next step for the case study presented in Section 6.3, looking for advantageous driving modes in the powertrains that were generated using this works method.

In many approaches presented in the literature review in Section 2, functions are used within the synthesis process, e.g. [27, 47, 50, 34]. Functions are often defined as a combination of a noun and a verb, e.g. "increase pressure", that relate to the energy-, material- and signal-flow in the system [10]. In [51] a functional basis is presented that gives a standardized terminology for the definition of functions. Functions were initially introduced to support designers in a solution-neutral approach to a design task and to overcome their bias towards already known solutions. This work is not using functions for the synthesis. The port-based modeling of components implicitly includes their functions in the sense of converting inputs to outputs and the system boundary reflects the technical system's overall function that is the "overall relationship between the inputs and the outputs of a plant, machine or assembly" [10]. Further, the mapping from functions to components is often conducted by port-matching, i.e. selecting components that reproduce the inputs and outputs of the associated function, e.g. in [27, 77]. In the method presented in this thesis, this step can be considered redundant, since the components are used to connect the ports of the system boundary equivalent to modeling an overall functions with elementary functions. If the required functionality of the product can be provided by the generated concepts is evaluated using the generated constraint satisfaction problems and the simulation models in the sense of a test for functionality [21]. Therefore, the explicit use of functions is not needed in this work and would not render any advantage.

7.1 Contributions

Where the mentioned literature always only tackles one part, either the topological synthesis, or variable assignment and simulation, this work combines topological synthesis,

generation of variable assignments and finding the most opportune variable assignment using optimization and simulation in one seamless approach. In order to achieve this, three different generic model transformations are introduced. Starting with the metamodel in the object-oriented graph-based concept modeling framework (cf. Section 4.1) a first-order logic describing the solution space of topologies. For the CMGs that are generated in the topological synthesis two further model transformations are provided: i) to a constraint satisfaction problem describing the variable solution space for the design represented by the CMG, and ii) to a bond graph-based simulation model that can be used to evaluate the solution at hand. In order to assure that the metamodel only contains compatible partial simulation models for each element, modeling guidelines are introduced.

In the following paragraphs this work's impact is elaborated guided by the refined research questions defined in Section 2.5:

> 1. How to formally encapsulate a task specification and engineering knowledge without using rules for conceptual design to allow the generation and automated behavioral simulation of the solution space using advanced search methods?

The answer to this question, provided by this work, is the metamodel presented in Section 4.1. The object-oriented aspect of this engineering knowledge repository allow including compatibility constraints modeled in the port configuration and the definition of relations between these interfaces by equations and simulation models. The graph-based aspects, envisioning components as possible nodes in a graph support the accessibility for human designers. This accessibility is also facilitated by the fact that the designer does not have to define rules to encapsulate the engineering knowledge. In order to make advanced search methods available, this work uses the solving techniques that are accessible for Boolean satisfiability problems. The first version of DPLL-algorithm was developed in 1958 and has since then been further developed [115]. Torlak [85] developed a way to map relational logic to Boolean satisfiability problems. This work presents a generic way to translate engineering design tasks that are modeled in the metamodel to a first-order logic description and, by that, allow using the advanced algorithms to solve Boolean satisfiability problems. A further advantage is that future developments in SAT and SAT-solving can be easily used since the actual solving process is independent from the modeling of the design task.

> 2. How to implement and execute model transformations to support automating conceptual design?

Being able to transform generated models in CDS allows adding viewpoints to such models that were not intended, or are not available in the original representation. To facilitate the method presented in this thesis, model transformations between the metamodel and first order logic, generated solutions and constraint satisfaction problems, and generated solutions and bond graph-based simulation models were developed. The overall method constitutes a way to include such transformations into CDS to support the generation and evaluation step of the general CDS framework by Cagan et al. [18]. In this

work, the transformations enable the synthesis of concept topologies using the algorithms available for solving Boolean satisfiability problems, the generation variable assignments using constraint satisfaction and the behavioral simulation of the generated concepts.

3. How to enable a mapping between abstract components and simulation models describing their behavior for computational design synthesis?

As described in Section 2.5, the success of design automation in VLSI design is highly due to the independence of the generic components that are used to design new chip designs [8]. Therefore, the concepts used there cannot easily be transferred to solving engineering design tasks where different components highly influence each other. This work introduces a modeling guideline to allow a mapping between abstract components and corresponding partial simulation models. Characteristics of bond graphs are used to assure the compatibility of these models. This step allows the evaluation of generated concepts without being constrained to predefined simulation models and narrow fields of application.

7.2 Limitations & Future Work

Several challenges exist using this method. The metamodel definition has to be carried out carefully. Although the inclusion of declarative knowledge in a metamodel, as used in this work, is more intuitive than in rules or rule sequences and can, therefore, more easily be taught to designers from industry, the formulation of an engineering task as a PMP is not trivial. A further refinement of the port and element hierarchies is often needed to prevent insufficient solutions. Additionally, modeling the partial simulation models for the system boundary and the different components requires in-depth knowledge of the included domains. However, considering the method in an industrial context, the different components would be modeled and designed by specialist departments. The development department, where the final synthesis is carried out, provides the port hierarchy to ensure the compatibility of the different components and defines the system boundary for the development task that is to be tackled. Applying the method enables the development department to generate and evaluate the solution space for the specific task, instead of only investigating 5-20 different concepts, which is usual manual development processes [16]. However, this application is only an imaginary application. Future work should include user studies in an industrial context to investigate the modeling effort and support provided by the method.

Another issue is the calculation time. Since SAT problems are CPU-intensive, the calculation time for bigger scopes becomes unbearable. In this work's case studies the actual time per scope evaluation was limited due to the batch system working on Brutus[17]. Although a lot of the small scopes could be evaluated for a time limit of four hours in the second powertrain case study (cf. Section 6.3), most of the scopes were stopped at the time limit. However, during the work on this case study, the time limit of four hours proved to be a good compromise between waiting time on the server and solution space

[17] http://www.brutus.ethz.ch/

exploration. But, this means, that the complete exploration of the solution space cannot be guaranteed. Apart from the solving times regarding the SAT problem, the solving times for the CSP and the time to run the simulation are also to be considered. For the second powertrain case study (cf. Section 6.3) the generation of 10000 variable assignments need between 5 seconds and more than 160 hours CPU-time on Brutus. Applying parallel solvers and several instances of the solving process are used to reduce the actual time needed for solving. The optimization of the 97 different powertrain concepts needed in total approximately 395 hours which results in on average four hours for 500 iteration for each powertrain concept.

Future work has to consider strategies to reduce the total application time of the method. One possibility is to implement a better scope prediction to eliminate the time spent searching scopes with no solutions. Another possibility is to break large engineering tasks into smaller subtasks with, therefore, smaller scopes. Additionally, future work should consider a generic way to find reasonable settings for the different parameters of the method: i) the number of iterations for simulated annealing, ii) the number of variable assignments to generate, iii) the amplifier k for the optimization. The number of iterations directly influences the application time, since every simulation run, i.e. calculation of the objective function value, takes a certain amount of time depending on the complexity of the simulation model. The time expense for the generation of the variable assignments becomes linearly larger with each additional variable assignments. The influence of the amplifier k was already described in Section 4.4.

Another factor that influences the needed calculation time is the number of operation points that is defined in the system boundary. The following example data was revealed using the powertrain CMG presented in Figure 57 on page 88. Table 8 shows the times needed to generate 10000 variable assignments with different number of operation points and shows that the needed time increases with the number of operation points. However, the number of operation points also influences the number of iterations needed in simulated annealing. Figure 65 shows the convergence of the objective function value for a powertrain averaged over each ten runs of the optimization using simulated annealing with different starting points. As can be seen, having defined 13 operation points, the simulated annealing process starts in a more advantageous position. Only after about 80 iterations, the average cumulated error considering no operation points reaches the initial value of the average cumulated error considering 13 operation points.

Table 8: The influence of the number of operation points on the solving of the CSP for an electric powertrain with nine design variables and 10000 assignments to be generated.

Number of operation points [-]	Solving time on Brutus [s]
0	10
7	2355
13	3807

Implementing a stopping criterion that stops simulated annealing after n iterations with no change of the objective function value could be a possibility to reduce the application

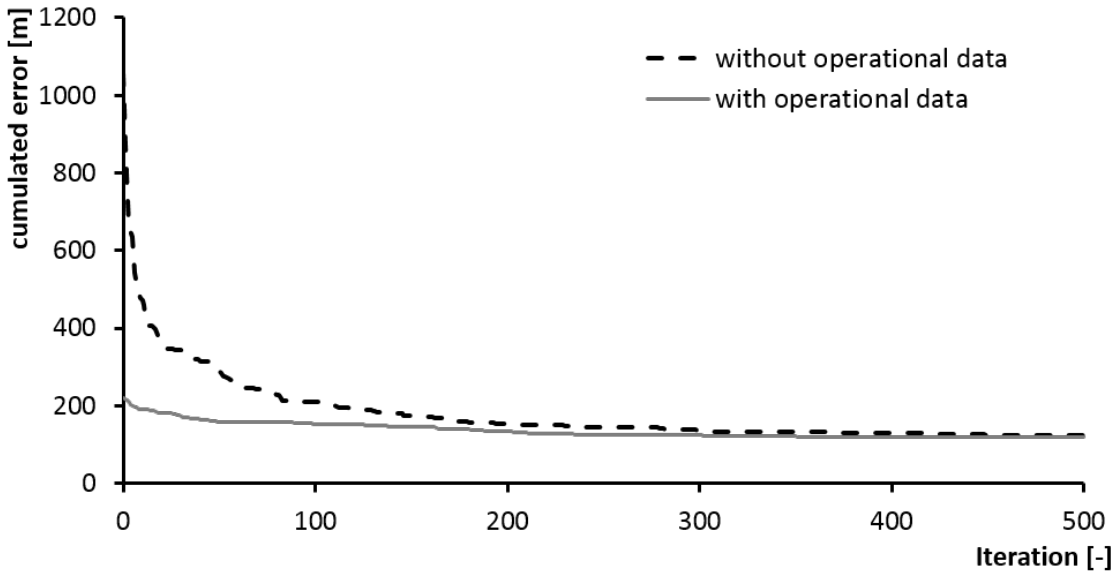


Figure 65: The average convergence of the ten runs of the simulated annealing process for a powertrain concept, once having defined none and once having defined 13 different operation points.

time. To investigate the impact of such criterion on application time and solution quality, a study on the example powertrain CMG mentioned above (cf. Figure 57) is conducted, including variable assignments generated with and without operation points.

The results of the study are shown in Figure 66. The average solution quality q_{av} is calculated following Eq. 71 with r_{n_i} being the objective function value when reaching the stopping criteria in run i, r_{m_i} being the objective function value after performing the maximum number of iterations for run i, and z being the number of runs with different starting points. The average time savings t_{av} are calculated using Eq. 72, with k_{n_i} being the total number of iterations performed in run i given the stopping criteria n and m being the maximum number of allowed iterations.

$$q_{av} = \frac{1}{z} \sum_{i=1}^{z} \left(1 - \frac{r_{n_i} - r_{m_i}}{r_{m_i}} \right) \tag{71}$$

$$t_{av} = \frac{1}{z} \sum_{i=1}^{z} \frac{m - k_{n_i}}{m} \tag{72}$$

The results of the study with z being 10, m being 500, and $n \in \{25, 50, 75, 100, 125, 150\}$ are shown in Figure 66. As can be seen, the implementation of a stopping criterion enables substantial time savings, especially for a small n. However, the result quality q_{av} decreases the smaller n is chosen. Investigating the influence of including operational points in the generation of variable assignments the following can be concluded. Whereas the time savings are similar whether operation points are included or not, the solution quality is consequently higher, especially for a small n. For instance for $n = 125$, the time savings with and without operational points are about 30 % (32.7 % with; 28.6 % without) but the result quality with operational data is at 99.6 % whereas the one without

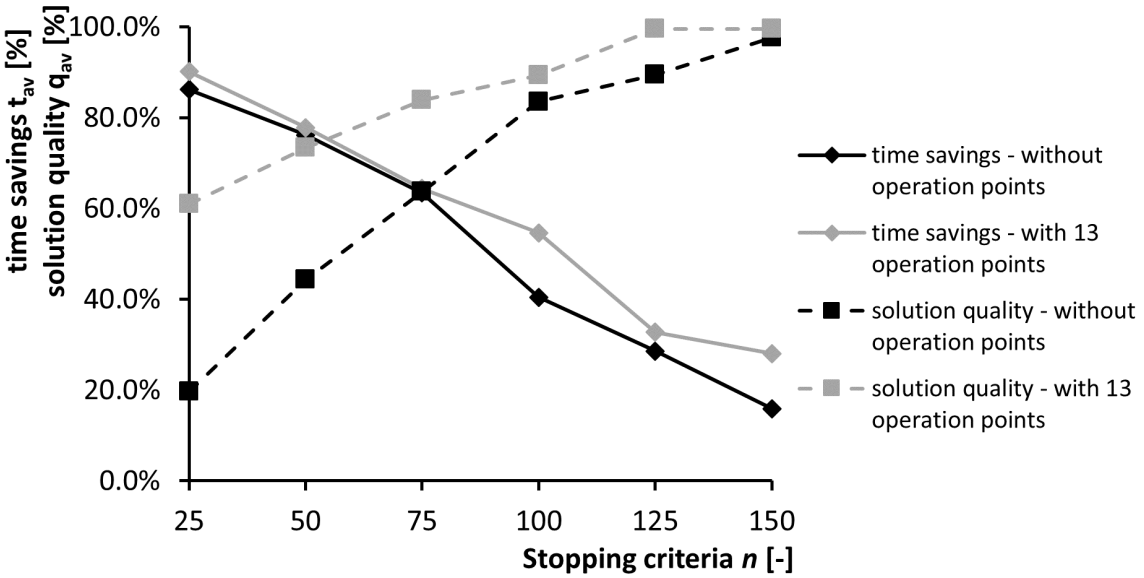


Figure 66: The average time savings and result quality (each averaged over ten runs) over the stopping criterion n given a maximum number of 500 iterations.

operational data is only 89.4 %. However, this study was only conducted on one powertrain example and the general validity of the findings has yet to be confirmed using a representative set of the results from the case studies.

If the observations are confirmed after generally including the stopping criterion in the method, future work should consider developing heuristics to automatically set the parameters of the method. To do that, the metamodel and the generated CMGs should be analyzed in order to find the most opportune combination of parameters that reduces the application time while maintaining a reasonable quality of the results.

Another approach that should be investigated is a hybrid approach using this method in combination with a graph grammar. Graph grammar rules can be generated using this work's topological synthesis approach by generating replacements for single components or connections. An example process is shown in Figure 67. Here, two example replacements for an internal combustion engine are generated by inverting the internal combustion engine's port configuration and including it in a system boundary. The topologies that are generated for this system boundary can be used to replace an internal combustion engine component in a CMG. The rules that result can be used to systematically evolve an initial model, which is generated for the design tasks initial system boundary. Due to being able to simulate the concepts, the effect of rule applications on the objective function value can be evaluated. Advantageous rules [116] and rule application sequences [117] can be identified and used to guide the generation process. This approach could combine the advantage of this work's method, i.e. that the designer is released from defining the rules manually, with the advantages of graph grammars, i.e. computationally inexpensive graph transformations and advanced search algorithms.

In this work, simulated annealing is used. As introduced in Section 3.4, simulated annealing [89] is a stochastic optimization method and was chosen because it showed

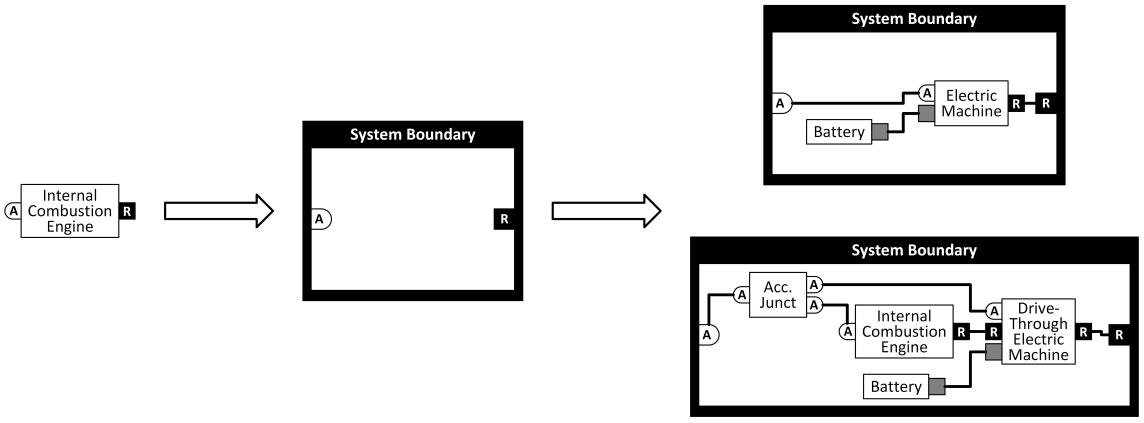


Figure 67: Example process for generating two example replacements for a combustion engine component using the metamodel from Section 6.3

its potential in a wide range of synthesis applications and the variables do not have to be encoded in a fixed representation for the method application. The method presented in this thesis could also work with other optimization methods. The only restriction is, that these methods have to allow the identification of the next variable assignment to be evaluated in the generated set of variable assignments. Despite this, more advanced optimization methods should be investigated to reduce the number of needed iterations and to avoid local optima and, therefore, to improve the results. One possibility is pattern search [118]. Here, the surrounding of the initial variable assignment are investigated at a certain distance and the next iterations are pursued in the direction that showed promise in the initial investigation. When the surroundings contain no more opportune assignments, the distance is decreased and the search continued. Another possibility is the covariance matrix adaptation evolution strategy (CMA-ES) [119]. In this method, starting at an initial variable assignment, variable assignments are investigated in a multivariate normal distribution around the initial assignment. A percentage of the best found assignments is used to average a new mean value for the normal distribution. The dependencies between the variables are embodied in a covariance matrix. This matrix is updated on every iteration to increase the possibility of pursuing previously successful search directions.

Besides looking for alternative optimization methods, two more limitations should be resolved. The first one is the current restriction to single-objective optimization. Making multi-objective optimization available would enable a better evaluation of the generated concepts. The second limitation is the almost exclusive use of continuous numerical variables. Whereas methods exist to solve mixed constraint satisfaction problem [71], the optimization method used in this work relies on the Euclidean distance between the variable assignments. This only works limited for variables in an integer domain, but not for other discrete variables. This issue should be resolved in future work.

Another research field that will be investigated is overcoming the limitation of only being able to generate and evaluate energy- and signal-based systems. As shown in Section 6.1, the synthesis method used in this work allows the free definition of ports. The general approach of combining partial simulation models, per se, works with any declarative modeling paradigm for simulation. Modelica, for instance, allows the user to define any

connector not being limited to different forms of energy and signals. However, modeling guidelines similar to the ones presented in this work for bond graphs have to be developed to assure that the generated concepts can be simulated in the end.

Design automation not only considers conceptual design but also other phases of the product development process. As mentioned before, this method is a big step in automating conceptual design, but it is dependent on the results from the planning and task clarification, i.e. the requirements list [10]. In order to automate requirements engineering, new challenges will have to be faced. A formal description of requirements is needed, which then allows to automatically refine to bring the the requirements to the needed level of granularity. One possibility could be adapting target cascading methods, as, for instance, introduced by Kim et al. [120] for automotive vehicle design. Further, to make it more accessible, natural language processing should be included as, for instance, presented in [121] in the domain of software requirements.

8 Conclusion

The gap between available information and the importance of the decisions makes conceptual design a crucial phase of every product development process. The trend towards higher product complexity incorporating different engineering domains even magnifies this gap. But for all that, the computational support in conceptual design is still sparse and uncommon.

In Section 4, this thesis presents a seamless automated computational design synthesis approach for the generation and evaluation of solution spaces for conceptual design tasks. The object-oriented graph-based framework provides designer with the possibility to encapsulate task specifications and engineering knowledge in the metamodel. Using the embedded knowledge, alternative concept topologies are generated using first-order logic descriptions and Boolean satisfiability. Assignments for the design variables are generated using constraint satisfaction and the optimal assignment is the generated set is determined using simulated annealing, evaluating the concepts with automatically generated simulation models.

This thesis intends to help to resolve the issues one through four raised in Section 1 and, by that, make CDS more attractive for engineering practice. The first challenge is the often overwhelming number of design alternatives that is presented to the user without sufficient evaluation. To this issue, this thesis contributes a new method to generate and evaluate solution spaces for engineering tasks. The mapping to simulation models allows the performance evaluation of the concepts in the actual task, giving designers a means to efficiently select promising concepts for embodiment design.

The second challenge states the difficulty of knowledge formalization and the lack of support for designers doing it. In this work's method, the designer is only concerned with the definition of the metamodel. In contrast to, for instance, rule-based approaches, there is no knowledge needed about the methods operators and their execution. The designer only defines the available components and the system boundary, the rest of the method is designed to work fully automatically. To support designers developing partial simulation models for the different component types, modeling guidelines are presented to assure a well defined causality in the generated simulation models.

The third challenge is the limitation of viewpoints towards a concept due to the limited use of different knowledge and concept representations. Towards this challenge, this thesis contributes generic model transformations between the object-oriented graph based modeling framework and i) first-order logic, ii) constraint satisfaction problems, and iii) bond graph-based simulation models. Each representation is used to enhance the completeness of the concept definition to support the designer with as much knowledge as possible to base his or her decision making on.

To address the fourth challenge, the shortfall of synthesis systems that scale up to the complexity of the tasks in engineering practice, in Section 6, four case studies are presented that aim at three different levels of complexity (II, III, and IV). This shows the ap-

plicability of the method for the design of technical system ranging from sub-assemblies (printhead drives in Section 6.4) over more complex machines (automotive powertrains in Section 6.2 and 6.3) to plants (chemical process engineering in Section 6.1). The metamodels provided for these case study further provide a good start and example application for further research projects.

The main issues that remain for future work are means to reduce the calculation time. These can include strategies to subdivide design tasks, more effective optimization algorithms or an automated model transformation towards a set of graph rules that could be used in a guided search instead of exhaustive generation. Further, ways should be investigated to aim at further automating design process in the direction of planning and task clarification as well as in the direction of embodiment design.

With regards to the conceptual design phase according to Pahl et al. [10] (see Figure 4 on page 5), this thesis proposes a method that allows automating most parts of conceptual design. Especially when envisioning using the method over several projects, the metamodel is predefined and the designers' task is merely the definition of the system boundary. Applying the method generates a solution space including concept alternatives, each with a performance evaluation. From these alternatives, the designer can select the most promising concepts for the evaluation of economic criteria and after that, has a good foundation to support the decisions towards one or more principle solutions.

References

[1] C. Münzer, B. Helms, and K. Shea. Automated Parametric Design Synthesis Using Graph Grammars and Constraint Solving. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Chicago, USA, August 2012.

[2] C. Münzer, B. Helms, and K. Shea. Solving Design Tasks in Engineering Using Object-Oriented Graph-Based Representations and Boolean Satisfiability. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Portland, USA, August 2013.

[3] C. Münzer, B. Helms, and K. Shea. Automatically Transforming Object-Oriented Graph-Based Representations Into Boolean Satisfiability Problems for Computational Design Synthesis. *Journal of Mechanical Design*, 135(110):101001–1 – 13, 2013.

[4] C. Münzer and K. Shea. A Simulation-based CDS Approach: Automated Generation of Simulation Models Based From Generated Concept Model Graphs. In *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Boston, USA, August 2015.

[5] B. Kruse, C. Münzer, W. Wölkl, A. Canedo, and K. Shea. A Model-based Functional Modeling and Library Approach for Mechatronic Systems in SysML. In *ASME 2012 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Chicago, USA, August 2012.

[6] B. Kruse, C. Münzer, S. Wölkl, A. Canedo, and K. Shea. Workflow and Modeling Conventions for Function and Product Structure Modeling of Mechatronic Systems in SysML using Libraries. In R. Scheidl and B. Jakoby, editors, *The 13th Mechatronics Forum International Conference Proceedings*, volume 2 of *Advances in mechatronics*, pages 506–513, Linz, 2012. Trauner Verlag.

[7] C.L. Dym and D.C. Brown. *Engineering Design*. Cambridge University Press, second edition, 2012. Cambridge Books Online.

[8] D.E. Whitney. Why mechanical design cannot be like vlsi design. *Research in Engineering Design*, 8(3):125–138, 1996.

[9] G.E Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, 1965.

[10] G. Pahl, W. Beitz, J. Feldhusen, and K.-H. Grote. *Konstruktionslehre*. Springer Verlag, Berlin, Germany, 7 edition, 2007.

[11] Dassault Systèmes. CATIA V5. http://www.3ds.com/products-services/catia/, 2015. last accessed on August 10th, 2015.

[12] Siemens PLM. NX 10. http://www.plm.automation.siemens.com/en_us/products/nx/index.shtml, 2015. last accessed on August 10th, 2015.

[13] R.R. Young. *The Requirements Engineering Handbook*. Artech House, Norwood, MA, USA, 2004.

[14] K. Ehrlenspiel, A. Kiewert, and U. Lindemann. *Cost-Efficient Design*. Springer-Verlag, Berlin, Germany, 5 edition, 2007.

[15] K. Ehrlenspiel. *Integrierte Produktentwicklung*. Carl Hanser Verlag GmbH & Co. KG, München, Germany, 4 edition, 2009.

[16] Y.-C. Liu, T. Bligh, and A. Chakrabarti. Towards an 'ideal' approach for concept generation. *Design Studies*, 24(4):341–355, 2003.

[17] E.K. Antonsson and J. Cagan, editors. *Formal Engineering Design Synthesis*. Cambridge University Press, Cambridge, UK, 2 edition, 2005.

[18] J. Cagan, M.I. Campbell, S. Finger, and T. Tomiyama. A Framework for Computational Design Synthesis: Model and Applications. *Journal of Computing and Information Science in Engineering*, 5(3):171 – 181, 2005.

[19] A. Chakrabarti, K. Shea, R. Stone, J. Cagan, M.I. Campbell, N.V. Hernandez, and K.L. Wood. Computer-based design synthesis research: an overview. *Journal of Computing and Information Science in Engineering*, 11(2):021003–1–021003–10, 2011.

[20] D.E. Knuth. Computer science and mathematics: How a new discipline presently interacts with an old one, and what we may expect in the future. *American Scientist*, 61(6):pp. 707–713, 1973.

[21] S. Thomke and T. Fujimoto. The Effect of 'Front-Loading' Problem-Solving on Product Development Performance. *Journal of Product Innovation Management*, 17:128 – 142, 2000.

[22] VDI-Fachbereich Produktentwicklung und Mechatronik. VDI 2221 - Methodik zum Entwickeln und Konstruieren technischer Systeme und Produkte, 1993.

[23] C. Königseder. *A Methodology for Supporting Design Grammar Development and Application in Computational Design Synthesis*. accepted PhD-Thesis, ETH Zurich, Zurich, Switzerland, 2015.

[24] A. Falkner, A. Haselböck, G. Schenner, and H. Schreiner. Modeling and solving technical product configuration problems. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25(2):115–129, 2011.

[25] B. Fachbach. Anforderungen an die Konzeptphase aus Sicht der OEMs - Ergebnisse einer Befragung. In *4. Grazer Symposium Virtuelles Fahrzeug*, Graz, Austria, May 2011.

[26] W.E. Eder. Design modeling-a design science approach (and why does industry not use it?). *Journal of Engineering Design*, 9(4):355–371, 1998.

[27] B. Helms and K. Shea. Computational Synthesis of Product Concepts based on Generalized Graph Grammars. *Journal of Mechanical Design*, 134(2):021008, 2012.

[28] S. Mittal and F. Frayman. Towards a generic model of configuraton tasks. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 89, pages 1395–1401, 1989.

[29] D.C. Brown. Design configuring. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 12(4):301 – 305, 1998.

[30] W.P. Birmingham, A.P. Gupta, and D.P. Siewiorek. The micon system for computer design. *Micro, IEEE*, 9(5):61–67, 1989.

[31] S. Mittal, C.L. Dym, and M. Morjaria. Pride: An expert system for the design of paper handling systems. *Computer*, 19(7):102–114, 1986.

[32] D. Sabin and R. Weigel. Product configuration frameworks - a survey. *IEEE Intelligent Systems*, 12(4):42 – 49, 1998.

[33] F. van Harmelen, V. Lifschitz, and B. Porter, editors. *Handbook of Knowledge Representation*. Elsevier, Amsterdam, The Netherlands, 1 edition, 2008.

[34] L.C. Schmidt and J. Cagan. GGREADA: A graph grammar-based machine design algorithm. *Research in Engineering Design*, 9(4):195 – 213, 1997.

[35] L.C. Schmidt, H. Shetty, and S.C. Chase. A graph grammar approach for structure synthesis of mechanisms. *Journal of Mechanical Design*, 122(4):371 – 376, 2000.

[36] R. Alber and S. Rudolph. "43" - A generic approach for engineering design grammars. In *AAAI Spring Symposium "Computational Synthesis"*, pages AAAI Technical Report SS–03–02, Stanford, CA, USA, March 2003.

[37] J. Schaefer and S. Rudolph. Satellite design by design grammars. *Aerospace Science and Technology*, 9:81–91, 2005.

[38] B. Landes and S. Rudolph. Aircraft cabin architectures including tolerancing using a graph-based design language in UML. In *Deutscher Luft- und Raumfahrtkongress 2011*, Bremen, Germany, September 2011.

[39] A.C. Starling and K. Shea. A parallel grammar for simulation-driven mechanical design synthesis. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Long Beach, USA, September 2005.

[40] F. Bolognini, A.A. Seshia, and K. Shea. Exploring the Application of Multidomain Simulation-based Computational Synthesis Methods in MEMS Design. In *International Conference on Engineering Design 2007 - ICED'07*, Paris, France, August 2007. Design Society.

[41] Y. Lin, K. Shea, J. Pears, and A. Johnson. A method and software tool for automated gearbox synthesis. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, San Diego, USA, August - September 2009.

[42] T. Kurtoglu and M.I. Campbell. Automated synthesis of electromechanical design configurations from empirical analysis of function to form mapping. *Journal of Engineering Design*, 20(1):83–104, 2009.

[43] M.I. Campbell, R. Rai, and T. Kurtoglu. A stochastic tree-search algorithm for generative grammars. *Journal of Computing and Information Science in Engineering*, 12(3):031006–1–11, 2012.

[44] T. Stanković, M. Štorga, K. Shea, and D. Marjanović. Formal modelling of technical processes and technical process synthesis. *Journal of Engineering Design*, iFirst, 2012.

[45] R.S. Hutcheson, R.L.Jr. Jordan, R.B. Stone, J.P. Terpenny, and X. Chang. Application of a genetic algorithm to concept variant selection. In *ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Philadelphia, USA, September 2006.

[46] D.F. Wyatt, D.C. Wynn, and J. Clarkson. A computational method to support product architecture design. In *ASME 2009 International Mechanical Engineering Congress and Exposition*, Lake Buena Vista, USA, September 2009.

[47] D.F. Wyatt, D.C. Wynn, J.P. Jarrett, and P.J. Clarkson. Supporting product architecture design using computational design synthesis with network structure constraints. *Research in Engineering Design*, 23(1):17–52, 2012.

[48] J. Rihtaršič, R. Žavbi, and J. Duhovnik. SOPHY - Tool for structure synthesis of conceptual technical systems. In *11th International Design Conference - DESIGN'10*, Dubrovnik, Croatia, May 2010.

[49] M.-L. Moullec, M. Bouissou, M. Jankovic, J.-C. Bocquet, F. Réquillard, O. Maas, and O. Forgeot. Toward system architecture generation and performances assessment under uncertainty using bayesian networks. *Journal of Mechanical Design*, 135(4):041002–1 – 13, 2013.

[50] Y. Jin and W. Li. Design Concept Generation: A Hierarchical Coevolutionary Approach. *Journal of Mechanical Design*, 129(10):1012–0–1012–10, 2007.

[51] J. Hirtz, R.B. Stone, D.A. McAdams, S. Szykman, and K.L. Wood. A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts. *NIST Technical Note*, 1447:1–43, 2002.

[52] B. Helms, J. Schultheiss, and K. Shea. Automated Assignment of Physical Effects to Functions Using Ports Based on Bond Graphs. *accepted for Journal of Mechanical Design*, 2013.

[53] B. Helms. *A Rule-based and Model-based Knowledge Representation for Computational Design Synthesis*. Technische Universität München, Garching, Germany, 2013.

[54] H.A. Simon. *The sciences of the artificial*. The MIT Press, Cambridge, MA, USA, 1969.

[55] D. Braha. Design-as-satisfiability: a new approach to automated synthesis. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 15(5):385 – 399, 2001.

[56] A. Hatchuel and B. Weil. A new approach on innovative design: an introduction to C-K theory. In *International Conference on Engineering Design - ICED'03*, Stockholm, Sweden, August 2003.

[57] O. Shai, Y. Reich, A. Hatchuel, and E. Subrahmanian. Creativity theories and scientific discovery: a study of C-K theory and infused design. In *International Conference on Engineering Design - ICED'09*, Stanford, USA, August 2009.

[58] L. Hendriks and A. Osman. A method for design reasoning using logic: from semantic tableaux to design tableaux. In *International Conference on Engineering Design - ICED'11*, Copenhagen, Denmark, August 2011.

[59] L. Hendriks and A.O. Kazakci. A design assistant architecture based on design tableaux. In *International Design Conference - DESIGN 2012*, Dubrovnik, Croatia, May 2012.

[60] A.A. Kerzhner. *Using Logic-based Approaches to Explore System Architectures for Systems Engineering*. Georgia Institute of Technology, Atlanta, USA, 2012.

[61] Object Management Group (OMG). OMG Systems Modeling Language (OMG SysML^{TM}). http://www.omg.org/spec/SysML/1.3/, 2012. last accessed on July 30th, 2015.

[62] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, Amsterdam and Oxford, 1 edition, 2006.

[63] M. Stumptner, A. Haselböck, and G. Friedrich. Cocos-a tool for constraint-based, dynamic configuration. In *Artificial Intelligence for Applications, 1994., Proceedings of the Tenth Conference on*, pages 373–380, Mar 1994.

[64] M. Stumptner and A. Haselböck. A generative constraint formalism for configuration problems. In Pietro Torasso, editor, *Advances in Artificial Intelligence*, volume 728 of *Lecture Notes in Computer Science*, pages 302–313. Springer Berlin Heidelberg, 1993.

[65] L. Purvis and P. Pu. Adaptation using constraint satisfaction techniques. *Case-Based Reasoning Research and Development*, 1010:289–300, 1995.

[66] D. Sabin and E.C. Freuder. Configuration as composite constraint satisfaction. In *Proceedings of the Artificial Intelligence and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[67] E. Silvas, T. Hofman, A. Serebrenik, and M. Steinbuch. Functional and cost-based automatic generator for hybrid vehicles topologies. *Mechatronics, IEEE/ASME Transactions on*, PP(99):1–12, 2015.

[68] P.A. Yvars, P. Lafon, and L. Zimmer. Optimization of mechanical system: contribution of constraint satisfaction method. In *International Conference on Computers & Industrial Engineering 2009*, pages 1379–1384. IEEE, 2009.

[69] E. Gelle, B.V. Faltings, D.E. Clément, and Smith I.F.C. Constraint Satisfaction Methods for Applications in Engineering. In *Engineering with Computers, 16 (2000) 2*, pages 81–95. Springer, 2000.

[70] E. Gelle, B. V. Faltings, and I.F.C. Smith. Structural Engineering Design Support by Constraint Satisfaction. In *Artificial Intelligence in Design 2000*, pages 311–331. Kluwer Academic Publishers, 2000.

[71] E. Gelle and B. Faltings. Solving Mixed and Conditional Constraint Satisfaction Problems. *Constraints*, 8(2):107–141, April 2003.

[72] A.A. Shah, C.J.J. Paredis, R. Burkhart, and D. Schaefer. Combining Mathematical Programming and SysML for Automated Component Sizing of Hydraulic Systems. In *ASME 2010 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Montreal, Quebec, Canada, 2010.

[73] U. Sellgren. *Simulation-Driven Design - Motives, Means, and Opportunities*. KTH Stockholm, Stockholm, Sweden, 1999.

[74] C.-S.N. Shiau, N. Kaushal, C.T. Hendrickson, S.B. Peterson, J.F. Whitacre, and J.J. Michalek. Optimal plug-in hybrid electric vehicle design and allocation for minimum life cycle cost, petroleum consumption, and greenhouse gas emissions. *Journal of Mechanical Design*, 132(9):091013–1 – 11, 2010.

[75] C.-S.N. Shiau and J.J. Michalek. Global optimization of plug-in hybrid vehicle design and allocation to minimize life cycle greenhouse gas emissions. *Journal of Mechanical Design*, 133(8):084502–1 – 6, 2011.

[76] R. Hauffe, C. Samaras, and J.J. Michalek. Plug-in hybrid vehicle simulation: How battery weight and charging patterns impact cost, fuel consumption, and co2 emissions. In *ASME 2008 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, New York, USA, August 2008.

[77] A. Canedo and J.H. Richter. Architectural design space exploration of cyber-physical systems using the functional modeling compiler. In *24th CIRP Design Conference*, Milano, Italy, April 2014.

[78] A.E. Bayrak, Y. Ren, and P.Y. Papalambros. Design of hybrid-electric vehicle architectures using auto-generation of feasible driving modes. In *ASME 2013 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Portland, USA, August 2013.

[79] Z. Wu, M.I. Campbell, and B.R. Fernández. Bond graph based automated modeling for computer-aided design of dynamic systems. *Journal of Mechanical Design*, 130(4):041102–1–11, 2008.

[80] T. Johnson, A. Kerzhner, C.J.J. Paredis, and R. Burkhart. Integrating models and simulations of continuous dynamics into sysml. *Journal of Computing and Information Science in Engineering*, 12(1):011002–1–11, 2012.

[81] K. Saitou, K. Izui, S. Nishiwaki, and P. Papalambros. A survey of structural optimization in mechanical product development. *Journal of Computing and Information Science in Engineering*, 5(3):214–226, 2005.

[82] D.W. Hoffmann. *Theoretische Informatik*. Carl Hanser Verlag, München, Germany, 1 edition, 2009.

[83] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. *Computer Aided Verification*, pages 641–653, 2002.

[84] N. Eén and N. Sörensson. An extensible SAT-solver. In *11th international conference: Theory and Applications of Satisfiability Testing*, Guangzhou, China, May 2008.

[85] E. Torlak. *A Constraint Solver for software engineering: finding models and cores of large relational specifications*. The MIT Press, Cambridge, USA, 2009.

[86] D. Jackson. *Software abstractions: logic, language and analysis*. The MIT Press, Cambridge, USA, 2 edition, 2012.

[87] W. Borutzky. Bond graph modelling and simulation of multidisciplinary systems - An introduction. *Simulation Modelling Practice and Theory*, 17(1):3 – 21, 2009.

[88] W. Borutzky. *Bond graph methodology: development and analysis of multidisciplinary dynamic system models*. Springer Verlag, Berlin, Germany, 2010.

[89] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[90] S.P. Brooks and B.J.T. Morgan. Optimization using simulated annealing. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 44(2):pp. 241–257, 1995.

[91] J. Cagan and K. Kotovsky. Simulated annealing and the generation of the objective function: a model of learning during problem solving. *Computational Intelligence*, 13(4):534–581, 1997.

[92] B. Raphael and I.F.C. Smith. *Engineering informatics: Fundamentals of computer-aided engineering*. John Wiley & Sons, West Sussex, UK, 2013.

[93] V. Hubka and W.E. Eder. *Theory of Technical Systems*. Springer Verlag, Berlin, Germany, 1 edition, 1988.

[94] G.L. Snavely and P.Y. Papalambros. Abstraction as a configuration design methodology. *Advances in Design Automation*, 1:297–305, 1993.

[95] International Organization for Standardization. Information technology - Metadata registries (MDR)-Part 1: Framework, ISO/IEC 11179-1, 2004.

[96] W3C OWL Working Group. OWL web ontology language. http://www.w3.org/TR/owl-guide/, 2004. last accessed on July 30th, 2015.

[97] E. Jakumeit, S. Buchwald, and M. Kroll. GrGen.NET. *International Journal on Software Tools for Technology Transfer*, 12(3):263–271, 2010.

[98] D. Jackson. Alloy: a lightweight object modeling language notation. *ACM Transactions on Software Engineering and Methodology*, 11:256–290, 2002.

[99] N. Sörensson and N. Eén. Minisat v1.13-a sat solver with conflict-clause minimization. In *SAT Competition 2005*, St Andrews, Scotland, June 2005.

[100] D. Le Berre and A. Parrain. The Sat4j library, release 2.2 system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[101] E. Goldberg and Y. Novikov. BerkMin: a fast and robust SAT-solver. *Discrete Applied Mathematics*, 155(12):1549–1561, 2007.

[102] M. Moskewicz, C. Madiagn, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *39th Design Automation Conference*, Las Vegas, USA, June 2001.

[103] Gecode Team. Gecode: Generic constraint development environment, 2015. Available from `http://www.gecode.org`.

[104] Siemens PLM. LMS Imagine.Lab Amesim. http://www.plm.automation.siemens.com/en_us/products/lms/imagine-lab/amesim/, 2015. last accessed on August 16th, 2015.

[105] L.R. Petzold. A description of dassl: A differential/algebraic system solver. In *Proc. IMACS World Congress*, pages 430–432, 1982.

[106] L.R. Petzold. Automatic selection of methods for solving stiff and nonstiff systems of ordinary differential equations. *SIAM journal on scientific and statistical computing*, 4(1):136–148, 1983.

[107] C. Schulte, G. Tack, and M.Z. Lagerkvist. Modeling. In Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist, editors, *Modeling and Programming with Gecode*. 2015. Corresponds to Gecode 4.4.0.

[108] L.T. Biegler, I.E. Grossmann, and A.W. Westerberg. *Systematic Methods of Chemical Process Design*. Prentice Hall PTR, New Jersey, USA, 1 edition, 1997.

[109] EuropeanUnion. 70/220/EWG - RICHTLINIE DES RATES vom 20. März 1970 zur Angleichung der Rechtsvorschriften der Mitgliedstaaten über Massnahmen gegen die Verunreinigung der Luft durch Emissionen von Kraftfahrzeugen, 1970.

[110] BDEW - Bundesverband der Energie und Wasserwirtschaft e.V. Datenerhebung 2012 - Bundesmix 2012. https://www.bdew.de/internet.nsf/id/1E7BD75876AE0D08C1257823003ED8C4/$file/13-08-21_Bundesmix%202012%20Stromkennzeichnung.pdf, 2012. last accessed on July 30th, 2015.

[111] ADAC e.V. Autodaten & Kosten - Toyota Prius 1.5 Hybrid (06 - 09). https://www.adac.de/infotestrat/autodatenbank/detail.aspx?KFZID=205680, 2015. last accessed on July 30th, 2015.

[112] M. Aebischer, Y. Beer, F. Grossenbacher, A. Nieland, L. Rüttimann, and A. Vogel. 3D-Printing Raptype Building the Future - Endbericht Fokusprojekt Raptype, 2014. unpublished focus project report, ETH Zürich.

[113] H.P. Geering. *Optimal control with engineering applications*. Springer Verlag, Berlin, Germany, 113 edition, 2007.

[114] T.A. Johansen and T.I. Fossen. Control allocation - a survey. *Automatica*, 49(5):1087 – 1103, 2013.

[115] J. Franco and J. Martin. A history of satisfiability. *Handbook of Satisfiability*, 185:3–74, 2009.

[116] C. Königseder and K. Shea. Systematic Rule Analysis of Generative Design Grammars. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28(3):227 – 238, 2014.

[117] C. Königseder and K. Shea. Improving Generative Grammar Development and Application Through Network Analysis Techniques. In *International Conference on Engineering Design (ICED)*, Milano, Italy, August 2015.

[118] R. Hooke and T.A. Jeeves. "direct search" solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[119] N. Hansen, S.D. Muller, and P. Koumoutsakos. Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[120] H.M. Kim, D.G. Rideout, P.Y. Papalambros, and J.L. Stein. Analytical target cascading in automotive vehicle design. *Journal of Mechanical Design*, 125(3):481–489, 2003.

[121] V. Ambriola and V. Gervasi. Processing natural language requirements. In *Automated Software Engineering, 1997. Proceedings., 12th IEEE International Conference*, pages 36–45. IEEE, 1997.