Decision Diagrams for the Verification of Consistency in Automotive Product Data

Entscheidungsdiagramme zur Verifikation der Datenkonsistenz in der Automobilproduktion

Der Technischen Fakultät
der Friedrich-Alexander-Universität
Erlangen-Nürnberg
zur
Erlangung des Doktorgrades

DOKTOR-INGENIEUR

vorgelegt von
Rüdiger Berndt
aus Neuendettelsau
Als Dissertation genehmigt
von der Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg

Tag der mündlichen Prüfung: 20.01.2016
Vorsitzender des Promotionsorgans: Prof. Dr. Peter Greil

Gutachter:
Prof. Dr.-Ing. habil. Reinhard German
Prof. Dr.-Ing. habil. Richard Lenz
To my father,
who taught me some of the most important things about life
but, unfortunately, died much too early.
(1938-1998)
Acknowledgements

I would like to express my thanks to my Ph. D. supervisor and head of the chair for computer networks and communication systems of the Friedrich-Alexander-Universität Erlangen-Nürnberg: Professor Reinhard German. Although being entrusted with numerous tasks regarding lectures, scientific work, and management, he always had enough time to support me during this project—due to his considerateness and experience, he always gave the right advice, which was important for the project’s success.

I want to thank Professor Richard Lenz (chair for data management), Professor Thomas Moor (chair for control theory), and Professor Michael Glaß (chair for hardware-software-co-design) for their kind support during the final steps of this project.

Many thanks to all of my colleagues at the chair for a great time, including the interesting and amusing discussions in the coffee breaks. Particularly, I would like to thank Peter Bazan and Dr. Kai-Steffen Hielscher for their support regarding the development of the verification approach and the according publications.

I owe my thanks to Dr. Martin Lukasiewycz for his readiness to help me with his expertise in the field of satisfiability solvers.

The interdisciplinary work and publications together with Dr. Juliane Blechinger (chair for data management) and Dr. Florian Risch (chair for factory automation and production systems) enriched my knowledge of the corresponding domains and inspired new ideas, thank you!

I want to thank the AUDI AG for sponsoring this project, and all colleagues for their kind support—among others my supervisors Dr. Thomas Stöckel, Frank Siefert, and Jörn Elberfeld as well as Martin Riederer, Andrea Albrecht, and Maximilian Vogel. In particular, I want to thank Dirk Zitterell for the productive teamwork regarding the meta-rules approach and the corresponding publication.

Last but not least I want to thank my friends, my family, Jessica, and my daughter Lilli.
Abstract

The automotive manufacturing domain of these days is faced with several challenges. According to Maslow’s hierarchy of needs (American psychologist, 1908-1970), the desire for individuality will steadily increase along with the growing prosperity of people. This is why the concept of mass customization has been evolving since the 1990s, i.e. “producing goods and services to meet individual customer’s needs with near mass production efficiency” [69]. Adding to this comes the progressing globalization which involves new competitors and saturated markets, especially in Europe and Northern America. In turn, car manufacturers have not only continuously developed their product range but also reduced the time to market (TTM).

To achieve this, the simultaneous engineering (SE) paradigm for the development cycles of cars was adopted. On that score, well-performing flows of information are considered to be among the fundamental prerequisites. In order to implement them, a high degree of standardization is required. As a side effect, many companies struggling for markets suffered a loss of flexibility—but flexibility is essential to efficiently react to both rapidly changing framework conditions and the fast technological progress.

The multitude of different concepts established by various business units is usually reflected by a high complexity within the underlying data structure. Due to this and the limited technical options regarding the design of information technology (IT) systems that prevailed in the past, in many cases a truly diverse and interlaced system topology has been evolving. While the flows of information keep getting more complex and non-transparent, the consolidation of IT system architecture is becoming ever more difficult. Nevertheless, the interdependence of business processes and IT systems is continuously growing as well. As a result, the influence of data w.r.t. decision-making and process coordination is steadily rising, thus leading to a new relevance of data quality (DQ). This is why diverse methods and management approaches to ensure high DQ were developed. But due to the application-oriented nature of many DQ problems, these methods often remained abstract and theoretical.

Faced with these circumstances, the AUDI AG started a project to replace the paper-based human-machine interfaces (HMIs) controlling the assembly process of cars by an IT system. Its big advantage is given by the reduction of complexity for the operators. Still a problem is that the new system heavily relies on correct input data. Being aware of that, the AUDI AG established this doctoral research study to ensure high quality of the involved data.

In the course of this thesis, a new technique is introduced focusing on the verification of consistency within the different data sources. For this, a formal model of the data including
a set of consistency requirements will be developed. Based on that, *multi-valued decision diagrams* (MDDs) will be deployed, on the one hand to encode large sets of configurations (also: customer orders), and, on the other hand, to efficiently carry out set operations which are the basis for the verification process. The verification method requires an MDD-based representation of the feasible design space, i.e. the set of all valid configurations. But the compilation of the configuration problems of highly customizable cars leads to high memory consumption—determined by the number of nodes. The latter, in turn, can be influenced by the variable ordering, i.e. the arrangement of levels within the diagram. Therefore, both well-known and new variable ordering approaches will be applied to keep the size of the MDDs practicable.

In addition to that, during the compilation of the configuration problems, the sequence of processing the constraints determines the dynamics of memory consumption. In order to keep the growth of memory consumption low, two new algorithms will be introduced. It will be demonstrated that the combined application of the presented methods allows to construct the MDD representation of the feasible design space of all investigated products.

The results of the MDD-based verification technique will be validated by using a SAT-based technique (from ‘Boolean satisfiability problem’) which also allows to compare the performance of both approaches.
Kurzfassung


Zu deren Verwirklichung wird während der Entwicklung von neuen Produkten der ‘simultaneous engineering’ (SE)-Ansatz verfolgt, d.h. “verteilte gleichzeitige Entwicklung” [Übersetzung Autor]. In diesem Zusammenhang sind gut funktionierende Informationsflüsse als wesentliche Voraussetzung zu nennen. Um diese zu realisieren, ist jedoch ein hoher Grad an Standardisierung erforderlich. Im Zuge dessen erleiden viele Firmen, die um Markanteile kämpfen, einen Flexibilitätsverlust. Flexibilität ist jedoch unentbehrlich angesichts der sich stetig verändernden Rahmenbedingungen und des schnellen technologischen Fortschritts.


Vor diesem Hintergrund startete die AUDI AG ein Projekt mit dem Ziel, die papierbasierten Mensch-Maschine-Schnittstellen zur Steuerung der Montage durch ein IT-basiertes (engl. information technology, IT) System zu ersetzen. Der große Vorteil liegt in der Komplexitätsreduktion für die Werker, allerdings ist das neue System auf korrekte Eingabedaten angewiesen. Angesichts der Tatsachen hat die AUDI AG das vorliegende Forschungsprojekt zur Absicherung der DQ ins Leben gerufen.

Im Rahmen dieser Forschungsarbeit wird ein neues Verfahren zur Verifikation von

Während der Konstruktion des zulässigen Entwurfsraums wird die Dynamik des Speicherbedarfs durch die Reihenfolge der Constraint-Verarbeitung bestimmt. Um das Anwachsen des Speicherbedarfs zu verringern, werden zwei neue Methoden vorgestellt. Es wird gezeigt, dass durch die kombinierte Anwendung der neuen Methoden die MDD-Darstellung des zulässigen Entwurfsraumes aller untersuchten Produkte erfolgreich konstruiert werden kann.

Die Ergebnisse der MDD-basierten Verifikation werden mithilfe eines SAT-Solvers validiert, außerdem werden die Laufzeiten der beiden Ansätze verglichen.
## Contents

Acknowledgements iii  
Abstract v  
Kurzfassung vii  
Contents ix  

1 Introduction 1  
1.1 Motivation 1  
1.2 Research Focus 3  
1.3 Structure of the Thesis 5  

2 Automotive Manufacturing 7  
2.1 The Product Development Process 8  
2.1.1 The Definition Period 9  
2.1.2 The Development Period 10  
2.2 Simultaneous Engineering 12  
2.2.1 Fundamentals 13  
2.2.2 Hazards and Pitfalls 15  
2.3 The Production Process 16  
2.3.1 Assembly 18  
2.4 Summary 19  

3 Data Quality 21  
3.1 Fundamentals 22  
3.1.1 Data Errors 22  
3.2 Data Quality Dimensions 25  
3.2.1 Categories of Dimensions 27  
3.3 Data Quality Management 29  
3.4 Summary 30  

4 Formal Model of the Data 33  
4.1 Fundamentals 33  
4.2 Configuration Data and Customer Orders 37  
4.2.1 Design Space 38  
4.2.2 Feasible Design Space 39  
4.3 Bill of Materials 40  
4.3.1 Parts and Constraints 42  
4.3.2 Part Types and Composite Constraints 42  
4.4 Assembly Plan 44  
4.5 Mapping 46  
4.6 Consistency Requirements 47  
4.6.1 Consistency of the Configuration Data 48
1 Introduction

1.1 Motivation

In the last decades, paper has been used as carrier of the information controlling the assembly process of cars. We distinguish the Bauschein, bearing information on the customer order, from the Prüfkarte, which is used to trace the state of construction; this record is important for the maintenance of the quality control loop. Both media accompany the cars throughout the assembly process and, accordingly, are not specifically tailored to the information requirements of different workstations. Each Prüfkarte contains the quality measures and test steps for all possible equipment features—even though customer orders might contain only a subset of equipment features. Each Bauschein, likewise, contains the codes of all selected equipment features—even if not all of them are necessary for the identification of the assembly steps to execute and the parts to assemble regarding a single workstation. Hence, the operator(s) of each workstation must carry out an interpretation process to map the codes of the Bauschein into the required parts as well as into the necessary assembly and test steps.

During the last decades, not only the complexity of cars themselves, observable for example in terms of an increasing number of available equipment features, but also the amount of information on the Bauschein and the Prüfkarte has been continuously growing. In many cases, this led to a phenomenon known as cognitive burden. Consequently, both the risk of performing an erroneous interpretation of the codes as well as the time for the documentation of the state of construction have increased, too. Furthermore, the medium paper is characterized by inflexibility and marked with a number of shortcomings, including the limitation of space or the environmental balance. Given these disadvantages, paper is no longer appropriate for the control of the assembly of highly customizable cars in the 21st century.

Against this background, the AUDI AG has launched the project Elektronische Wagenbegleitkarte (eWBK) to replace both paper media by an IT system (see Figure 1.1). The necessary input data, for a given product and an assembly line, is composed of

- the bill of materials (BOM),
- the assembly plan (AP), and
- the customer orders—derived from the configuration data (CD).

These input data are combined within the eWBK system, which performs a mapping to determine the required tasks and parts for each workstation and customer order, respectively.
1. Introduction

(a) Scope of the pilot project *Elektronische Wagenbegleitkarte* (eWBK)

- Reduction of complexity
- Enhancement of process and product quality
- Continuous electronic monitoring of the test results and the state of construction
- Efficient control cycles
- Indication of errors and problem spots

(b) *eBauschein* and *eQRK*—from the german word ‘*Qualitätsregelkreis*’ (QRK)

Figure 1.1: Replacing paper by an IT system

(see Figures 1.2 and 4.6). By doing so, the complicated interpretation processes for the operators will become dispensable in the future and the cognitive burden will be lessened, thus the overall quality of the assembly process is expected to increase. The eWBK system does not only minimize the risk of misinterpretation but also boosts flexibility; considering, for example, late-time changes of customer orders or a more efficient regulation of repair and restoring work. In addition to that, firstly, considerable amounts of paper can be saved and, secondly, the cycle times can be reduced.

The new system was road-tested within a very stable and high-quality assembly process environment—the assembly line of the *Audi A3* in Ingolstadt—and can be considered as pilot project for the VW Group. Numerous issues were handled during the implementation phase, including the assessment and adaption of the processes providing the input data, the acquisition and roll-out of hardware equipment for the workstations, or the planning and realization of the network access and power supply.
1.2. Research Focus

Regardless of all substantial modifications of the productive environment, this project also comprised a non-negligible strategical and political component. Of course, on the one hand, the risk of misinterpretation is reduced, but, on the other hand, the chances for operators to detect wrong assembly instructions are also constricted. Consequently, errors within the input data might cause a systematic deviation of the assembly process; for example, all customer orders to be delivered to Japan might happen to get equipped with a version of the navigation system missing the adequate language option. This is why the focus of this research project is on the quality of the input data and methods to ensure their correctness.

1.2 Research Focus

The basis to control the manufacture of customized products is given by constraints. Constraints within the CD are used to express whether or not combinations of equipment features are feasible, i.e. selectable by customers. Constraints connected to a part within the BOM or an assembly step within the AP are used to characterize customer orders whose production require this part or assembly step, respectively. The domains of the variables that are used to encode customer orders or to specify constraints are finite sets—for example, the car’s
color which can either be blue, red, green, black, or white, for example. This structural property is reflected by the formal model established in the course of this thesis, allowing to map constraints to sets of product configurations. The structure of these sets, in turn, is suited to be efficiently represented by MDDs.

As part of the data’s formal model, a collection of consistency requirements is developed—motivated by the interdependencies among the input data and the mapping of the eWBK system. The verification of these requirements is based upon the feasible design space, i.e. the set of valid configurations. Therefore, the corresponding MDD representation must incorporate the information represented by the constraints in the CD (also: knowledge compilation). The compilation of these constraints often causes an immense memory consumption due to the large number of nodes in the diagram. In order to reduce their number, the variable ordering—i.e. the arrangement of levels within the MDD—can be modified. However, finding an optimal variable ordering belongs to the class of NP-complete problems [12].

Methods to enhance the variable ordering can be classified into static and dynamic. While static approaches improve the variable ordering a priori, i.e. before the construction, dynamic methods modify the variable arrangement during or after the construction. Accordingly, dynamic methods benefit from knowledge of the actual number of nodes. In contrast to that, static approaches exploit knowledge of the problem domain. This is why the application of dynamic methods usually yields better results; but their application—based on an elementary swap operation of two adjacent levels within the MDD—causes considerably more computational effort compared to static methods. In this thesis, next to analyzing state-of-the-art static and dynamic variable ordering methods, a new static variable ordering approach based upon the mass-spring-relaxation (MSR) paradigm will be introduced. It will be demonstrated, that the MDD representation of the feasible design space can be constructed in reasonable time by applying the new static approach along with a combined application of dynamic methods.

The growth of the MDD’s memory consumption is not only influenced by the variable ordering but also by the constraint compilation sequence. When using an inadequate sequence, the size of intermediate MDDs might exceed the memory limitations during the construction. In this thesis, two new methods for the optimization of the constraint sequence will be introduced; both of them will be designed to be applied dynamically, i.e. during the construction. It will be demonstrated that their application causes an almost monotonic growth of the MDDs memory consumption.
1.3. Structure of the Thesis

Summing up, this thesis will focus on the following research topics:

1. Applicability of MDDs for feasible design space representations of real-world automotive configuration problems, including methods to enhance:
   a) the variable ordering
   b) the constraint compilation sequence.

2. Verification of consistency requirements of automotive product data using MDD-based feasible design space representations.

3. Validation of the MDD-based technique using a SAT-Solver, including performance evaluation w.r.t. the run-time.

1.3 Structure of the Thesis

The structure of this thesis is as follows: Chapter 2 will take a closer look at the manifold activities within the automotive manufacturing domain. This includes the product development process (PDP) because the input data is generated during its application. Since the SE paradigm is applied w.r.t. the development cycles of cars, Section 2.2 will introduce this principle—including a section on both core strategies as well as hazards and pitfalls. The new eWBK system controls the assembly process of cars. Therefore, Section 2.3 will briefly touch the production process with a special focus on the assembly.

In Chapter 3, the term DQ as well as fundamental concepts related with it will be introduced. These include an overview of both data quality dimensions and data quality management approaches. Section 3.4 provides a brief overview of the framework conditions of the eWBKs information flows and discusses generic countermeasures to overcome DQ problems.

In the course of Chapter 4, a formal model of each component of the input data will be developed (Sections 4.2 to 4.4). Section 4.5 will explain the mapping—i.e. the way of how these components are combined to derive the information on the screens of the eWBK system. The dependencies among the components of the input data are the basis for the development of consistency requirements which will be presented in Section 4.6.

Based on the formal model and the consistency requirements, Chapter 5 will introduce both the MDD-based technique and a model for the SAT-based technique. Its first section will focus on the term MDD—including fundamental programming paradigms of the MDD software. In Section 5.1.3, the construction of the feasible design space as well as the verification of the consistency requirements will be described. Finally, Section 5.2 will describe the model for the SAT-based approach.

Chapter 6 will present diverse methods to improve the performance of the MDD-based approach. It is subdivided into two sections: While Section 6.1 will introduce methods to
improve the variable ordering, Section 6.2 will introduce methods to improve the constraint compilation sequence.

In Chapter 7, the applicability of the MDD-based approach will be demonstrated w.r.t. six real-world automotive products, including more than 30 variants altogether. It will be shown that the size of the MDD remains practicable by combining the methods of Chapter 6. Finally, the performance of the MDD-based approach will be compared with the SAT-based approach.

The last chapter of this thesis will present a summary and indicate the relevance of its content for future work.
2 Automotive Manufacturing

Few inventions changed the face of the world like the automobile did (also: car, from the Latin word ‘carrus’ which means ‘wheeled vehicle’). Its etymological origin is given by the Greek word ‘autós’, i.e. ‘self’, and the Latin word ‘mobilis’, i.e. ‘movable’, and stands for ‘a vehicle that moves itself’. The hour of birth of modern cars, powered by combustion engines, is said to be the year 1885 when Carl Benz (1844-1929) built the Patent-Motorwagen Nummer 1. From then on a lot of things have changed; manifold developments and patents pave the car’s path through history, like the invention of automatic transmission (1921), airbags (1951), catalytic converters (1956), three-point seat belts (1959), or the steady evolution of propulsion technology. Since gasoline became the prime source of energy for combustion engines, cars henceforth have replaced classical horse-drawn vehicles due to several advantages: increased velocity, range, and power, among others. Aside from the technological progress, there have been numerous enhancements of the production and manufacturing processes as well; for instance, Henry Ford’s (1863-1947) invention of assembly lines for the manufacture of cars in 1913. It made the concept of mass production of automobiles feasible and increased the degree of affordability and availability. Later, given limited means of production after the second World War, the Toyota Production System (TPS) was developed and still keeps on evolving. It is based on the fundamental principles of Toyoda Sakichi (1867-1930); its primary goal, namely elimination of waste, is reached by continuously monitoring and improving the production and business processes. This principle is known as continuous improvement process (jap.: ‘Kaizen’), for further information see [30]. Looking at nowadays, globalization and the trends of modern society pose new challenges to car manufacturers. On the one hand, further competitors enter the market, and, on the other hand, both the European and Northern American market have almost reached saturation.

Therefore, many car manufacturers started to further develop their product range to generate additional growth (see Figure 2.1, a). Consequently, the efforts for the maintenance of the products’ data (see Figure 2.1, b) steadily increased. In order to enhance competitiveness, an even more efficient response to customer needs is required as well. These can be identified as a growing demand, firstly, for affordable and customizable products and, secondly, for further improvements regarding safety, environmental sustainability and comfort. Given these facts, not only the functionality and quality of products but also the efficiency of the involved business processes needs to be continuously improved. Figure 2.2 depicts a survey of various circumstances adding complexity to the business of automotive manufacturers.

All things considered, the complexity of cars has been steadily growing over time due to a number of reasons. Furthermore, the risk of misunderstandings and errors has increased, since the SE principle is applied during the PDP.
Therefore, this chapter takes a closer look at the manifold activities within the car manufacturing domain—to become aware of the complexity within the products, the data, and the business processes. Its content is organized as follows: Section 2.1 introduces the PDP. While Section 2.2 will focus on the SE paradigm, corresponding core strategies, and potential pitfalls, Section 2.3 will introduce the production process of cars focusing on the assembly.

2.1 The Product Development Process

The development of a car, from the initial idea to the first delivery, covers a period of approximately five years. During this time, many different activities are performed, including the determination of the design, the creation and development of drawings and construction data, as well as the planning and preparation of the supply chain and the production process. Equally important is the management of manifold activities regarding the sales, marketing, human resources, and controlling business. For the purpose of supporting the complex coordination of all processes, a standardized process model—the PDP (see Figure 2.3)—has been developed. It subdivides the overall time into different periods, the accomplishment of each period’s essential objectives is marked by milestones (red squares). Throughout the PDP, all substantial results and the financial status are monitored and regularly reported to the relevant supervisory committees. The PDP is divided into a definition and a development period.

\[^{1}\text{In accordance with the PDP of the AUDI AG.}\]
2.1. The Product Development Process

Increasing number of products Expanding range of competencies Development of new equipment features Heterogeneous IT infrastructure Dynamics within global market Dynamics of legal requirements Varying political framework conditions Rapid technological progress High customer expectations

Figure 2.2: Internal and external circumstances adding complexity to the business of automotive manufacturers

2.1.1 The Definition Period

Discussing visions, concepts, and ideas of a new product involves all business units and officially initiates the development process, i.e. the project start (PS). This includes considerations of general technological aspects, marketing-related topics, as well as financial and organizational framework conditions. In order to successfully position the new product in the market, both customers’ expectations and competitors are comprehensively analyzed. Together with the brand and group strategies, this analysis yields the key characteristics of the product and allows to forecast the production volume. Meanwhile, the design department starts to develop several 2D drafts illustrating the exterior and interior of the new product. These drafts are gradually refined and finally complemented by true-to-scale 3D physical and virtual design models. By reaching the milestone project feasibility (PF), which marks the end of the definition period, the number of alternative design concepts is narrowed down to a limited selection.

Another key point of the definition period is the determination of an appropriate platform. A platform includes a set of standardized components—in particular for the vehicle’s body—that have been developed for the production of several outwardly distinct products. This strategy allows car manufacturers to cut development costs but also bears the risk of reducing the degree of differentiation.
Since the availability of diverse driver assistance systems has become an increasingly important selling point, the desirable electronic functions of the new product are defined next. With this in mind, a list displaying potential technological innovations is prepared. This list is complemented by the key indicators depicting environmental performance including weight, air drag coefficient, CO₂ emission, or noise generation. However, not only the product itself but also issues on marketing and sales-related strategies are topics of the definition stage. Furthermore, it is necessary to schedule the product’s stepwise initiation in various markets as well as to plan the availability of equipment features—like engines or gearboxes—during the product’s life cycle. Finally, both the production process and the supply chain are discussed, which also includes a suitability analysis of the available production sites.

As shown above, the result of the definition period is a catalog which responds to the requirements of the market by specifying the key characteristics of the new product. In addition to that, both a limited selection of designs and the framework conditions for the production and the acquisition processes are considered approved.

### 2.1.2 The Development Period

The PDP subdivides the development period, following after the definition period, into a period for the development of components and a period for the preparation of the serial production.

The assessment of the technical feasibility is based on both virtual and physical concept cars and is completed by reaching the milestone concept decision (CoD). At this time, the board of management officially appoints the designs to refine and the product’s key technologies to further pursue. Based on that, a comprehensive analysis of the modules’ and components’ reusability is conducted. This requires careful considerations, firstly, regarding
‘carry-over-parts’ (i.e. parts assembled in the preceding product\(^2\)) and, secondly, ‘common parts’ (i.e. parts yet mandatory for the production of other products), to determine the components which have to be redeveloped. Starting from then, the BOM will be continuously refined and enriched by various information on the product’s parts, including information on the time schedule, the suppliers, the weight, or the requested amount. Coupled with that, information on technical restrictions w.r.t. the combinability of components is gathered, encoded, and steadily updated within the CD. The project’s specification sheet is released when the milestone *design decision* (DD) is reached. Hereby, all requirements on quality, design, functionality, and costs are officially announced and the period for the development of components is completed.

The design, acquisition, and test of the tools and processes for the serial production are the focus of the activities within the period between the milestones DD and *launch* (La). Starting from then, all components and surfaces visible to customers are continuously monitored and checked with regard to their quality of appearance. For this purpose, virtual and physical *data control models* (DCMs) are established. Next to quality control, they support the development of production facilities including the

- toolmaking, i.e. the design and development of the tools for the pressing plant, or the
- layout of manufacturing cells for the body making, i.e. an almost fully automated, robot-assisted procedure to build the vehicle’s body by joining the structural elements.

Representing either single parts, partially assembled components, or even complete products, DCMs serve as input for simulation studies to reveal certain properties like, for example, the air drag coefficient. Coupled with that, the DCM approach supports the requirements engineering and change management process since the different versions of components can be easily compared. The official confirmation of the date for the *introduction to the market* (ITM) concludes the first part of the preparation period.

Starting from the launch until the *initial batch* (IB), all production facilities and production processes are examined and constantly optimized with the help of a pre-series production environment. By doing so, industrial engineers identify efficient work methods for the assembly process which are modeled and steadily maintained within the AP. The hereby fabricated vehicles are henceforth subject to extensive test routines and each revealed potential enhancement will be incorporated in the product. Meanwhile, the sales and marketing department develop the sales program—in terms of detailed time schedules for the initiation of the variants, equipment features, and decors within different markets. The corresponding information is stored and maintained within the CD. At the same time, further tasks, like the training of the operators and the briefing of the sales personnel, need to be planned and executed. Finally, when the readiness of production facilities has been officially

\(^2\)For example, currently the 9th generation of the *Audi A4* is developed.
approved, the start of the serial production (SOP) will be triggered. The PDP ends when the first cars are delivered to customers (ITM).

It is important to realize that the improvement of the product continues throughout the production period, starting from the SOP until the end of production (EOP). Usually, during this period and in the course of the development of further products, certain components will be redeveloped. Hence, such common parts or other improvements regarding functionality or look-and-feel are incorporated in the product.

### 2.2 Simultaneous Engineering

It is said to be the English naturalist Charles Darwin (1809-1882), founder of the theory of evolution, who claimed that “it is not the strongest of the species that survives, nor the most intelligent that survives—it is the one that is the most adaptable to change.”

Although this thought is related to the dynamics of evolution, it also reflects the dynamics of the global economic system that is characterized by competition. The ability to efficiently react to ever-changing circumstances and be ahead of competitors—regarding the development of innovative products—is considered an essential competitive factor for market participants. Unfortunately, the reduction of the lead development time (also: TTM) is made difficult by various factors, including the rapid technological progress and the increasing complexity of products (see Figure 2.2). In order to shorten the cycle times for the launch of new products, enterprises started to establish SE which, in principle, suggests the parallel execution of manifold activities during the PDP (see Figure 2.4). Hereby, cross-departmental teamwork and communication is induced—marked by the red arrows. This allows a continuous coordination and synchronization of the essential requirements throughout the development process and among
all involved business units. Consequently, an early integration of the demands of various stakeholders, including suppliers, customers, and internal process partners, reduces the number of misunderstandings and facilitates the detection of errors in a timely manner. In turn and unlike the classical sequential approach (also: over-the-wall), the application of SE is supposed to increase the product’s quality and, at the same time, to save development costs.

However, a successful application of SE heavily relies on efficient and error-free information flows; particularly when thinking of the added amount of cross-departmental communication. Every time information is exchanged via an interface, various errors may occur including misinterpretations and inconsistencies. Most of these errors cause additional burden—embodied in delays, rework cycles, and cost overrun. Consequently, companies applying SE are often unable to tap the full potential of time and cost savings.

2.2.1 Fundamentals

As shown in Figure 2.4, the general idea of SE is to parallelize the execution of processes. Therefore, the divisibility of a complex process into sub-processes is the principal prerequisite. This property is determined by the amount and nature of interdependencies between the corresponding sub-processes. Such dependencies are given, for example, by information or input data which is necessary to start or continue a certain process, but to obtain this specific input data, another process has to be executed first. Obviously, the interdependencies among sub-processes induce a sequence with regard to their execution.

Figure 2.5 shows two different patterns of progress w.r.t. mutual dependencies of processes. The example in the left depicts two completely independent processes: Process_A and Process_B. Apparently, both of them may be executed in a parallel manner without any limitation. The example in the right, contrarily, illustrates that Process_B depends on input that is obtained by executing Process_A. Therefore, the execution of Process_B

---

3Also the border between departments can be considered an interface.
cannot start until the according information has been acquired, at least partially. Since this input might be incomplete or fraught with uncertainty, a continuous communication between the process owners of Process\textsubscript{A} and Process\textsubscript{B} needs to be established to keep the information synchronized (red arrow). Due to the additional amount of communication, the execution time of both processes will increase (marked by the dark gray arrows). Generally speaking, the more efficient the synchronization mechanisms work, the more time will be saved compared to the sequential execution.

Since a car’s components and functions are strongly intertwined, the manifold processes executed while developing a car are characterized by numerous interdependencies, too. In addition, mapping responsibilities between business units, processes, and results induce further interfaces and complexity. Therefore, implementing powerful synchronization mechanisms is still a challenge, particularly when thinking of the heterogeneous IT system architecture.

The implementation of adequate standards has proven beneficial to efficiently integrate information. Besides, standardization does not only contribute to efficient flows of information but also facilitates the development of consistent designs for HMI's of IT systems. Typical subjects of standardization are given by the structure of the product (see Figure 4.4), the systems of concepts, the structure of the organization, or the business processes. The optimal level of regulation should balance the benefits, like increased transparency within data, along with the loss of flexibility. Hence, as a rule of thumb, standardization should only affect objects and activities whose character or structure do not vary from one project to another.

The performance of SE projects and the according cross-departmental communication is determined by various conditions. For example by the organizational structure of a company—affecting both the regularity of know-how exchange among all involved business units and the flexibility and creativity of the employees. Due to their interdisciplinary character, SE teams are considered a reasonable extension to traditional organizational structures. In the automotive domain the arrangement of SE teams often reflects the structure of the product—there are SE teams responsible for the engine, the bodywork, or the cockpit (see Figure 4.4). While the primary responsibility of an SE team refers to the coordination of the project, a further beneficial effect can be realized by steadily enriching and distributing the notable institutional know-how. SE teams usually consist of representatives of various business units. If necessary, they may also integrate external partners like suppliers or consultants. Hence, during decision-making, the solid expertise of all relevant fields will be directly at hand. SE teams, in essence, carry plenty of responsibility and have to handle big amounts of interpersonal communication—this is why team members should not only be specialists in their fields but also have sufficient behavioral competencies like conflict resolution, strategic thinking, and team building.

The SE concept is now widely adopted by many companies. Although various measures to improve its applicability have been proposed, there still exists potential to save further time. The next section will provide a brief overview of problems which may occur during the accomplishment of SE projects.
2.2.2 Hazards and Pitfalls

Structural properties, such as the divisibility of processes or the processes’ assignment to teams, determine the amount of required communication—in particular across team borders.

In the following, the term iteration is adopted according to [76]. An iteration is triggered whenever something has been modified; for example, the design of a component, a work method within the assembly process, or the functional principle of an assistance system—this information needs to be spread to all involved teams. Although it is difficult to anticipate the number of iterations, regardless of whether planned or unexpected, the division of a process should take the following issues into account:

• firstly, the expected number of changes,
• secondly, the amount of interdependence of the sub-processes,
• and, thirdly, the distribution of the sub-processes among the teams.

By way of example, let two processes with a high degree of mutual dependence be given: if possible, the responsibility for their execution should be assigned to the same team. This strategy allows to reduce the efforts of integrating results and coordinating the teams. Nevertheless, the division into sub-processes and the mapping of responsibility still primarily depends on the availability of sufficient competent human resources and the divisibility of the processes.

Next to structural circumstances, there is a non-negligible psychological component affecting the quality of interpersonal communication. In order to avoid potential trouble spots, the responsibility of each team and department must be clearly defined. By doing so, the emergence of diverse shortcomings like, for example, head monopolies, individual interests, or rivalry can be prevented. Coupled with that, an adequate appreciation of the work and know-how of colleagues who are not directly involved in an SE team will increase social cohesion and improve teamwork.

Another challenge is given by the management of digitized knowledge, including the design of IT systems and of the IT system architecture. Grown architectures are usually characterized by heterogeneity and comprise numerous interfaces. The exchange of information via such interfaces, regardless whether HMI or machine-to-machine interface (M2M), bears several risks, including misinterpretation, redundancy, and inconsistency. The corresponding data errors disturb the flow of information and cause additional effort. Besides that, poorly designed HMIs and insufficient transparency within the information flows do not only contribute to data defects but will also negatively affect the motivation of employees.

Summing up, applying SE leads to an increased amount of communication—satisfying as many conditions for efficient communication as possible, therefore, is considered the most important prerequisite. In the course of the next section, an insight into the production process of cars will be given. Since the eWBK system is settled within the assembly environment, Section 2.3.1 will focus on the assembly process.
2.3 The Production Process

The production planning and control (PPC, see Figure 2.6) constitutes the core business of producing industries, cf. [25, 29, 39, 40]. The Aachener Modell [40] distinguishes between core tasks: such as planning the production program, handling the demand for resources, or managing the supply chain—and a number of cross-sectional tasks: such as processing customer orders and monitoring objectives like stock, capacity utilization, or throughput time.

The production program represents the customer orders to be produced in the next production period. Based on that and the information within the BOM, the required parts and materials are identified. Most commonly, the acquisition is divided into in-house and external (cf. ‘real net output ratio’). The subsequent step deals with the optimization of the lot sizes and of the production sequence. There is a high degree of mutual dependence between a customer order on the one hand, and the workload of production facilities on the other hand, particularly when considering the workstations of the assembly line(s)\(^4\). Therefore, the production sequence should be optimized in such a way as to balance the

\(^4\)Sufficiently complex components (e.g. the cockpit) are assembled in extra pre-assembly lines. Later, all components are brought together in the final assembly line.
occupancy rate of workstations. The necessary information is contained in the AP in terms of execution times for the assembly steps—based upon the *methods-time measurement* (MTM) system, for example.

The main facilities of automotive production are given by the pressing plant, the body making, the paint shop, and the assembly. The production of engines and gearboxes is often carried out either in a separate location or outsourced—because foundries are required to cast the engine blocks, among others.

The car’s life cycle starts with the production of its body components in the pressing plant. This process is highly automated and its main expenses are caused by energy consumption and the maintenance of the equipment. Due to the enormous mass of the tools (> 20000 kg), a special lifting gear is necessary whenever they have to be exchanged. In the next step, during the almost fully automated body making, the body components are connected by using joining techniques like welding or clinching. For this purpose, numerous industrial robots are employed because, firstly, at this stage of production no variance needs to be covered—regarding the bodywork of only one specific product, and secondly, robots are characterized by high precision and speed. Afterwards, in the paint shop, the bodywork is subject to a number of specialized procedures, including electrophoretic dip painting or fully automated spray-paint lines. In the course of these electrochemical treatments, the bodywork is protected from corrosion and coated with varnish.
2.3.1 Assembly

The assembly process is an example for continuous flow production and the last step of a car’s production. From now on, the customer orders—including the selection of equipment features and decor—are assigned to the painted bodyworks. Before the actual assembly process starts, some components of the painted bodywork, like the doors or the tailgate, are disassembled and delivered to the corresponding pre-assembly lines. Assembly lines are made up of a number of workstations (approx. 400 workstations in the assembly lines of the *Audi A3*) which are organized in a sequence regarding the movement direction of the conveyer belt (see Figure 2.7). This sequence primarily depends on the parts to be assembled at each workstation considering their position in the vehicle—i.e. the sequence of assembly (cf. structure within the BOM in Section 4.3). For example, the wiring harness connecting all electrical components must be assembled before the interior lining is incorporated.

Once started, the assembly belt moves at a constant speed so that the operator(s) of each workstation are provided with a constant time interval \( t := \frac{\text{Cycle length}}{\text{Movement speed}} \) for the execution of the required assembly steps (approx. 1.5 minutes in the final assembly line of the *Audi A3*). Since customers may choose among numerous equipment features and decor, many parts are characterized by variance—considering all different versions of gear selectors, for example (cf. variance within the BOM in Section 4.3). This is why workstations are either stocked with all available variants and the operator has to select the correct part, or the parts are picked in an extra area and delivered to the corresponding workstation (cf. the concept of supermarkets and shopping baskets). In both cases, the operators choose the parts according to the customer order which is displayed on the paper-based *Bauschein*. Sometimes, the selection process is supported by computer-controlled systems (cf. pick-by-light) to minimize the error rate. Technical facilities of this kind are summarized within the Japanese term ‘poka yoke’ (‘error avoidance’), which is an essential principle of the TPS.

In order to prevent workstations from running out of stock, the internal suppliers deliver the parts *just in sequence* (JIS) or *just in time* (JIT), for example by adopting the ‘Kanban’ principle. So-called Kanban cards are used to indicate demand for certain parts at a workstation, and thus to control the supply chain from a production point of view.

Throughout the assembly process, and in particular after its completion, extensive test routines are carried out to ensure the high quality of the products; the corresponding results are recorded in the paper-based *Praßkarte*. According to the ‘jidoka’ principle (also: autonomation), operators can stop the conveyer belt whenever an error occurs during the assembly. When the error cannot be corrected in reasonable time, the car is redirected to the repair and restoring work. If possible, the error is indicated to the corresponding problem spot(s).

Finally, the last step of the car’s production focuses on the preparation for delivery. Depending on the destination country, different means of transport are taken into account: e.g. train, ship, truck, or direct pickup by customers.
2.4 Summary

Starting from 1885, the car has been subject to an eventful evolution, and this evolution will continue due to the steady social change (cf. urbanization or the shift towards information society) on the one hand, and the development of various technologies on the other hand. Technological progress can be observed, for example, in terms of the number of assistance systems and the required electronic control units (ECUs) within cars; starting from the 1980s—when only few ECUs for the engine management or the anti-skid system existed—until today—with up to 90 ECUs available for diverse safety and infotainment systems. These ever-changing circumstances do not only affect the cars themselves but also the requirements regarding the qualification of employees. Moreover, developing new systems and functions like autonomous driving, Car2X\(^5\), or the electrification of the power train impose further challenges to automotive manufacturers.

This chapter provided insight into two principal activity fields of car manufacturers: the PDP and the production process. Obviously, the manifoldness of the involved activities requires experts of most diverse qualifications. But, at the same time, both the increasing number of interdependencies of a car’s components and the application of SE during a product’s development presuppose interdisciplinary teamwork of all involved business units. Given these facts, efficient flows of information are considered an essential prerequisite because cross-departmental teamwork bears the risk of misinterpretations and errors. Therefore, the implementation of accurate flows of information is a key factor for future success.

The next chapter will provide an insight into current DQ research, including state-of-the-art methods to identify errors and to assess the quality of data. Such methods, in turn, are the basis for general management approaches allowing to improve DQ.

\(^5\)Implementation of an intelligent transportation system (ITS) by enabling communication among cars or between cars and infrastructure using modern WiFi or cellular-based communication systems.
3 Data Quality

In the 1960s, Gordon E. Moore (*1929), co-founder of Intel (Integrated electronics, founded 1968), predicted a doubling of transistors on integrated circuits every 1.5–2 years [48]. Moore’s law, proven true until today, is strongly related to the digital revolution (also: 3rd industrial revolution), which opened up entirely new possibilities in the field of information processing. Regarding performance, decisive factors like computation rate, size of memory, or bandwidth of communication media have been continuously improved due to the rapid technological progress during the last decades.

At the same time, the interdependence of IT systems and decision-making in most varied application fields has steadily grown as well. This shift towards an information society (cf. [73] for further information) is coupled with a phenomenon called information explosion challenging society to manage vast amounts of data. Moreover, data is often characterized by high complexity due to manifold applications and formats. As a rule of thumb: The higher the complexity within data and diversity among IT systems, the higher the error rate—and, accordingly, the lower the quality of data. Generally speaking, insufficient DQ is a source of misinformation which may cause bad decisions. High DQ, therefore, is considered essential for a company’s success [41]. Consequently, this topic has recently received more and more attention.

An example related to poor DQ (and/or system design) is given by the loss of the space probe ‘Mars Climate Orbiter’, which was launched by the National Aeronautics and Space Administration (NASA) in December 1998. Several months later, arriving at Mars in September 1999, the probe started to enter an elliptical orbit, but—after the spacecraft was supposed to leave the silent zone again—it was not possible to establish contact anymore. An investigative commission quickly revealed the reason for this: The navigation system brought the probe too close to Mars, therefore it disintegrated within its atmosphere. However, the actual source of errors was within the navigation system: While the NASA computed the impulse \( \vec{p} \) based on the SI-System\(^\text{1}\)—using Newton (N) as the unit for the force, the software for the navigation software was developed based upon the Imperial system—using Pound-force (lb) as the unit for the force. The difference in units (1 lb \( \approx \) 4.45 N) was not taken into account, and finally the whole system failed. This example, firstly, demonstrates the principle of small cause and large effect and, secondly, reveals a source of potential defects: inconsistency induced by different systems of measurement—other examples: the Julian and the Gregorian calendar, or different time zones.

This chapter provides an insight into current DQ research topics, including concepts,
models, and methods which were contributed to overcome DQ problems. It is subdivided into four sections: The following section introduces basic concepts including data errors and a classification model for data errors. The subsequent section provides an overview of data quality dimensions (DQDs). For this, in Section 3.2.1, also a taxonomy for their classification is introduced. Section 3.3 explains fundamentals of data quality management approaches. Finally, the last section summarizes the framework conditions w.r.t. the eWBK system and discusses generic measures to improve its DQ.

3.1 Fundamentals

Unfortunately, a generally valid definition of the term data quality is not available yet; the technical committee 184 of the International Organization for Standardization (ISO) is currently developing the standard: ISO8000, ‘Data quality’. Nevertheless, in order to establish a common interpretation of DQ, the definitions of the terms data and quality are introduced as follows:

**Definition 3.1: Data**
Re-interpretable representation of information in a formalized manner suitable for communication, interpretation, or processing [33].

**Definition 3.2: Quality**
The degree to which a set of inherent characteristics fulfills requirements [32].

Next to the above definition, quality is also paraphrased by ‘fitness for use’: accordingly, quality can be improved by incorporating the users’ viewpoints into the requirements analysis. This concept is also widely accepted within the DQ community where data quality means “data that are fit for the use by data consumers” [72].

By taking the definitions of data, quality as well as the concept of fitness for use into account, DQ can only be assessed if all DQ-related requirements are given. However, determining, analyzing, and adequately responding to all of them is a complex task, particularly when the set of actual consumers is unknown. This is aggravated by the fact that certain data, like the BOM, the AP, or the CD, are used in manifold applications and by various business units of a company (also: master data). In these cases, DQ problems will be marked by application-oriented views, which makes the search for appropriate countermeasures even more complicated.

3.1.1 Data Errors

Poor data quality is attributed to data errors: including wrong values, wrong formatted values, missing values, or contradictory values. An important pre-condition for the implementation of fault prevention mechanisms is the identification of the source(s) of dirty data. An adequate
3.1. Fundamentals

way to identify such trouble spots is to analyze and understand known errors; these insights, in turn, support the development and implementation of solutions for similar problems.

Preventing errors during the data's creation, like misspellings or misentries, is one primary goal to achieve high DQ—called 'quality at source'. Furthermore, it is obvious that the quality of a data source directly affects the quality of each connected downstream system; this phenomenon is paraphrased by garbage in, garbage out (GIGO) because usually computers process input data without questioning its correctness. In the worst case, nonsensical data will be reproduced. Hence, the more heterogeneous and interdependent an IT system architecture is, the more important the adoption of adequate data cleaning methods will become. “Data cleaning, also called data cleansing or scrubbing, deals with detecting and removing errors and inconsistencies from data in order to improve the quality of data” [55].

Focusing on data cleaning, Rahm and Do developed a taxonomy of major data quality problems (see Figure 3.1). At first, single-source problems are distinguished from multi-source problems. Each of the resulting classes is again subdivided into scheme level and instance level problems.

- Errors of the first class (single-source, scheme level, SS) are mainly caused by poor scheme design, for example by the lack of integrity constraints. In computer science, entity-relationship (ER) models can be used to describe the properties and relations within the data entities of an application. Improving the design of such models and incorporating appropriate integrity constraints will allow to prevent problems of this kind.

**Example 3.1: Single-source, scheme level**

A constraint could be introduced to ensure that a city cannot be the capital of more than one federal state—cf. uniqueness violation regarding the ‘Capital’ of ID 12 and 14 (Table 3.1). This would require to maintain two tables; one for each of the entities ‘Federal state’ and ‘City’.

![Figure 3.1: Classification of data quality problems according to Rahm et al. [55]](image-url)
Table 3.1: Data errors demonstrated by the example of the federal states of Germany

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Capital</th>
<th>Inhabitants [million]</th>
<th>Error class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Baden-Württemberg</td>
<td>Stuttgart</td>
<td>10.745</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Bayern</td>
<td>Nürnberg</td>
<td>12.539</td>
<td>SI</td>
</tr>
<tr>
<td>3</td>
<td>Berlin</td>
<td>Berlin</td>
<td>3.461</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Brandenburg</td>
<td>Potsdam</td>
<td>-2.503</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Bremen</td>
<td>Bremen</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>6</td>
<td>Hamburg</td>
<td>Hamburg</td>
<td>1.786</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Mecklenburg-Vorpommern</td>
<td>Schwerin</td>
<td>1.642</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Niedersachsen</td>
<td>Hannover</td>
<td>7.918</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Nordrhein-Westfalen</td>
<td>17.45</td>
<td>Düsseldorf</td>
<td>SI</td>
</tr>
<tr>
<td>10</td>
<td>Rheinland-Pfalz</td>
<td>Mainz</td>
<td>4.004</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Saarland</td>
<td>Saarbrücken</td>
<td>NULL</td>
<td>SI</td>
</tr>
<tr>
<td>12</td>
<td>Sachsen</td>
<td>Dresden</td>
<td>4.149</td>
<td>SS</td>
</tr>
<tr>
<td>13</td>
<td>Schleswig-Holstein</td>
<td>Kiel</td>
<td>2.843</td>
<td>SS, SI</td>
</tr>
<tr>
<td>14</td>
<td>Thüringen</td>
<td>Dresden</td>
<td>NULL</td>
<td>SS, SI</td>
</tr>
</tbody>
</table>

- The problems of the second class (single-source, instance level, SI), contrarily, cannot be prevented by enhancing the scheme design. Errors of this kind include missing values, misspellings, or misfielded values.

**Example 3.2: Single-source, instance level**
See Table 3.1 for the missing values of ‘Inhabitants’ (ID 5, 11 and 14), or the misspelling of ‘Capital’ (ID 2, correct: München), or the misfielded values of ‘Capital’ and ‘Inhabitants’ (ID 9).

According to the GIGO principle, the negative impact of single-source errors will be aggravated whenever information is composed of multiple data sources.

- Problems of the third class (multi-source, scheme level, MS) are often caused by naming and structural conflicts, such as homonyms: i.e. the same denotation for two or more different real-world objects; or synonyms: i.e. two or more different denotations for the same real-world object.

**Example 3.3: Multi-source, scheme level, Synonym**
Let three different data sources of the BOM be given; each of them denotes the unique identifier (UID) for the parts in a different way: ‘Part No.’, ‘Material No.’ and ‘Item No.’.
Further errors are induced by different representation schemes; regarding the usage of different data types and formats.

**Example 3.4: Multi-source, scheme level, Representation**

Let the gender of a person be attributed by a ‘0’ or a ‘1’ in the first data source, and by an ‘f’ or an ‘m’ in the second data source.

- Finally, the fourth class (multi-source, instance level, MI) contains problems which cannot be prevented, even if adequate schemes within the particular sources and adequate scheme translations for their integration are available. For example duplicates: Two or more records in a data set that refer to the same real-world object—popular from applications dealing with customer data.

Next to the taxonomy of Rahm and Do, further classification schemes have been proposed by Barateiro and Galhardas in [5], by Kim et al. in [35], or by Müller and Freytag in [49].

In general, awareness of data errors, potential trouble spots, and adequate countermeasures is indispensable considering the growing amount of data within many companies—often coupled with heterogeneous IT system architectures. However, the study of data errors only partially reflects the concept of DQ. There are further, more abstract properties which are not only determined by data values—such as the completeness or the reputation of a data source. In order to establish a common and more accurate understanding of DQ and to overcome the application-oriented nature of many DQ problems, the scientific community has developed the concept of DQDs, which will be introduced next.

### 3.2 Data Quality Dimensions

The Italian scientist Galileo Galilei (1564-1642) once proposed to “measure what is measurable, and make measurable what is not so”. Being faced with the challenge to make DQ measurable, scientists and practitioners contributed DQDs and metrics making DQ more tangible. This section introduces a selection of DQDs based upon the most common interpretations of literature. Still a problem is that denotation and interpretation of DQDs are not standardized; thus there are differences w.r.t. various authors².

In the article of Ballou and Pazer [4], the authors propose a data processing model to analyze the flows of information and the propagation of errors in information systems. This contribution is considered to be among the first research papers dealing with a definition of DQDs—namely of accuracy, timeliness, completeness, and consistency:

- The accuracy dimension (also: correctness) refers to the difference between the real-world value and the value reflected by the data stored in an IT system.

²See the comprehensive analysis and discussion of time-related dimensions in [60].
Considering that the claim for accuracy varies w.r.t. different decision-makings, this dimension should be perceived in a context-sensitive manner. Then, accuracy needs to be distinguished from correctness.

- The timeliness dimension captures whether a recorded value is up-to-date or not. It belongs to the family of time-related dimensions, which are controversially discussed in literature (cf. actuality or currency). Another interpretation of timeliness refers to the question of how quickly, i.e. ‘timely’, data is available for a specific task [60].

- The completeness dimension reveals whether data is missing. A more accurate interpretation is given by Pipino et al. [54] by distinguishing three different kinds of completeness: scheme completeness—i.e. “the degree to which entities and attributes are not missing from the scheme”, column completeness—refers to the number of missing values in a column of a table, and population completeness—i.e. the number of data entities compared with the cardinality of the real-world population.

- The consistency dimension refers to the equality of two or more different representations of a data value.

**Example 3.5: Consistency**

The distance between the cities of Nürnberg ‘Nuremberg’ (Franconia, Germany) and München ‘Munich’ (Bavaria, Germany) can be specified using either the Metric system: \( d_M = 170 \text{ kilometers} \), or the Imperial system: \( d_I = 106 \text{ miles} \). Both representations are considered consistent if the data values stick to the following equation: \( \frac{d_I}{d_M} = 0.62 \).

Another, more general, explanation is provided by Scannapieco et al. [60] where “the consistency dimension captures the violation of semantic rules defined over (a set of) data items”.

In order to exemplify the idea of making DQ measurable, a metric for the completeness dimension according to [28] will be introduced in the following.

**Example 3.6: Completeness, Metric**

If the value of an attribute in a database is denoted by \( \omega \), the function

\[
Q_C(\omega) := \begin{cases} 
0 & \text{if } \omega \text{ equals NULL} \\
1 & \text{else}
\end{cases}
\]  

(3.1)

captures whether or not a value for the attribute is specified. Based on that, the completeness of a set of attributes \( T := \{\omega_1, \ldots, \omega_k\} \) is evaluated by the arithmetic mean:

\[
Q_C(T) := \frac{\sum_{i=1}^{k} Q_C(\omega_i)}{k}
\]  

(3.2)

Extending the above equation by weighting factors allows to rate the importance of attributes individually. According to Table 3.1, the completeness of the column ‘Inhabitants’ is given
by \( \frac{14}{16} \approx 0.786 \). Regarding the overall number of entries ‘14’ and the actual number of federal states in Germany ‘16’, the completeness of the population reflected by Table 3.1 is given by \( \frac{14}{16} = 0.875 \).

Summing up, these four dimensions are discussed frequently in literature on DQ. However, there are numerous further dimensions which have been proposed (cf. [34]), for example:

- **Believability**: “the extent to which information is regarded as true and credible”.
- **Objectivity**: “the extent to which information is unbiased, unprejudiced, and impartial”.
- **Reputation**: “the extent to which information is highly regarded in terms of its source or content”.
- **Relevancy**: “the extent to which information is applicable and helpful for the task at hand”.
- **Value-added**: “the extent to which information is beneficial and provides advantages from its use”.
- **Accessibility**: “the extent to which information is available, or easily and quickly retrievable”.

As shown, some of the dimensions are interpreted differently: regarding the circumstances of a particular decision-making (timeliness or value-added), or the skill level of different user groups (accessibility). This is why there is no rule of thumb recommending to focus on a certain subset of DQDs—usually, the importance of dimensions and thus the prioritization of countermeasures can only be assessed by experts possessing sufficient domain knowledge.

### 3.2.1 Categories of Dimensions

In order to provide an overview of the growing set of DQDs, the DQ community developed various taxonomies for their classification, among others the *product and service performance model for information quality* (PSP/IQ)\(^3\) by Kahn et al. in [34], Redman’s classification in [56], or the approach given by Wand and Wang in [70].

Hereinafter, the framework of Wang and Strong [72] will be introduced. This empirical study follows the principle of fitness for use: the customers’—i.e. the users of data—perception of quality is the basis to assess the quality of the product—i.e. the quality of data. Hence, the authors suggest to “first collect data quality attributes from data consumers, and then collect importance ratings for these attributes and structure them into a hierarchical representation of data consumers’ data quality needs.”

\(^3\)Some authors distinguish data quality (DQ) from information quality (IQ); this thesis does not differentiate between both terms.
Table 3.2: DQ categories and dimensions according to Wang et al. [72]

<table>
<thead>
<tr>
<th>Category</th>
<th>DQ Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intrinsic</td>
<td>Believability</td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
</tr>
<tr>
<td></td>
<td>Objectivity</td>
</tr>
<tr>
<td></td>
<td>Reputation</td>
</tr>
<tr>
<td>Contextual</td>
<td>Value-added</td>
</tr>
<tr>
<td></td>
<td>Relevancy</td>
</tr>
<tr>
<td></td>
<td>Timeliness</td>
</tr>
<tr>
<td></td>
<td>Completeness</td>
</tr>
<tr>
<td></td>
<td>Appropriate amount of data</td>
</tr>
<tr>
<td>Representational</td>
<td>Interpretability</td>
</tr>
<tr>
<td></td>
<td>Ease of understanding</td>
</tr>
<tr>
<td></td>
<td>Representational consistency</td>
</tr>
<tr>
<td></td>
<td>Concise representation</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Accessibility</td>
</tr>
<tr>
<td></td>
<td>Access security</td>
</tr>
</tbody>
</table>

Therefore, the first part of the research involves a survey to collect attributes and ratings from data consumers to comprehend their “perspectives of data quality”. Altogether, 20 dimensions and their individual importance ratings were identified. During the subsequent step, categories and a mapping of dimensions to categories were revealed. Nevertheless, five dimensions were removed because, firstly, the subjects did not assign them in a consistent manner and, secondly, their importance rate was too low. The final result—a mapping of the 15 remaining dimensions to the four categories—is summarized in Table 3.2.

The intrinsic category comprises dimensions which are assessed on the basis of pure data. In contrast to that, further contextual information needs to be taken into account to rate the dimensions of the contextual category: like knowledge of the involved business processes or applications. The dimensions of the representational category do not assess the quality of the data but the quality of the datas’ presentation to the consumers. Finally, aspects of the datas’ access are reflected by accessibility category. Thus, “representational DQ and accessibility DQ emphasize the role of systems” [72].

In conclusion, the empirical model of Wang and Strong properly reflects the customer’s view on the variety of data quality aspects. In some cases, also the disagreement over both the importance and interpretation of dimensions is demonstrated. Nevertheless, it is comprehensible and thus suitably supports selecting appropriate quality dimensions w.r.t. a given DQ application.
3.3 Data Quality Management

The concept of quality assurance, in general, covers strategies and methods to ensure that products or processes will meet given quality requirements. This is why quality management strategies rather focus on application-independent process models than on concrete improvement activities. However, the generic idea behind these approaches is to continuously compare actual and target quality figures, and to launch countermeasures whenever the quality is insufficient. Well-known concepts of this application field include the iterative Plan-do-check-act (PDCA) cycle—which was made popular by Deming (1900-1993) and Shewart (1891-1967), the Define-measure-analyze-improve-control (DMAIC) process improvement method of Six Sigma ($6\sigma$), or the total quality management (TQM) approach:

**Definition 3.3: Total Quality Management**

Management philosophy and company practices that aim to harness the human and material resources of an organization in the most effective way to achieve the objectives of the organization [68].

A common and important philosophy behind the TQM approach is to let all members of an organization participate in the continuous improvement of the processes—to ensure the products’ quality and the long-term success of the company. However, as a precondition employees must develop a well-defined quality awareness.

For the purpose of achieving high DQ, basic ideas of known quality management approaches have been carried over to the field of information processing. The most popular among the resulting concepts is the total data quality management (TDQM) cycle by Wang [71] (see Figure 3.2). The approach is based on the idea of treating information as product(s) whose manufacture is supported by IT systems. The quality of such information products (IPs) can be determined, for example, based on DQDs and appropriate metrics. In general, DQ increases by iteratively executing four steps:

![Figure 3.2: TDQM cycle according to Wang [71]](image-url)
• Define: The first step is to identify the IPs and the way of how they are produced—i.e. the components of which they consist as well as the processes and IT systems that are involved in their ‘manufacture’. Equally important is to specify the quality requirements to be met by these IPs.

• Measure: The second step involves the development of metrics allowing to determine the quality of the IPs. Next to appropriate metrics, also the compliance with business rules—deployed to ensure high DQ—is systematically assessed.

• Analyze: The third step deals with the identification of trouble spots whenever the quality requirements are not fulfilled.

• Improve: The fourth step deals with the implementation of adequate countermeasures. By doing so, generic rules and procedures are tailored w.r.t. the specific circumstances of an identified trouble spot—regarding, for example, the ER model of an application dealing with federal states (cf. Example 3.1).

Further popular DQ management approaches are given by total information quality management (TIQM) \[22\] or AIM quality (AIMQ) \[38\]. However, a common characteristic of these approaches is abstractness. By not suggesting concrete measures, they remain applicable to many different areas of application. Still, abstractness may also be perceived an obstacle considering the adoption by companies. Usually, their DQ problems are highly application-oriented, hence rather concrete countermeasures are required.

### 3.4 Summary

DQ problems occur in many different fields of application. While online retailers strive for correct customer data, automotive manufacturers have to master the high dynamics of change, for example within the bill of materials. However, the application-oriented nature of DQ problems is not only determined by differences within applications and data, or by the heterogeneity of an IT system architecture but also by the organizational culture and the business processes. This makes the development of generic methods to overcome DQ problems even more complicated.

Figure 3.3 summarizes the framework conditions of the information flows regarding the input data of the EWBK system (cf. \[10, 58\]). In the final analysis, these circumstances do not only negatively affect the quality of data but also impede a successful implementation of adequate countermeasures.

Although no universal remedy yet exists, the causes for bad DQ are often of similar nature—considering the way of how data is generated within an IT system, for instance. Therefore, the relevant HMIs should be designed in such a way as to prevent faulty insertions (cf. ‘quality at source’) and all business processes generating or manipulating data should match the DQ-specific requirements. Requirements of this kind could be based upon
data quality dimensions, and in turn, appropriate metrics could be applied to validate the data’s conformance to such rules. In general, the amount of redundant data, and thus the synchronization efforts between IT systems via M2Ms, should be minimized to reduce the risk of asynchronous information.

Unfortunately, the implementation of DQ improvement measures is limited due to grown and heterogeneous IT system architectures. Such architectures are characterized by numerous interfaces; to reduce their number, systems should be modernized or even replaced. But, the rearrangement of IT system architectures involves high costs and risks—particularly within production environments.

Large companies are often faced with a manifold system of concepts; naturally, different divisions use different terms for the same real-world objects (cf. synonyms). However, employees have to understand each other when working together, particularly when considering SE projects and the related cross-departmental teamwork. This is why easily accessible and comprehensible standards should be developed.

A rather general circumstance which counteracts high DQ in organizations is revealed in [10] or [53]: awareness. Pierce et al. have compiled a list “regarding the obstacles that people [data quality professionals, editor’s note] perceived as most inhibiting data quality improvement in their organizations”. This list allows to conclude that the users, i.e. producers and consumers of data, need to be made aware of the relevancy of high DQ. Hence, next to the development and improvement of methods to solve classical DQ problems, like the detection and elimination of duplicates, the scientific community is faced with a number of further
topics: These include the development, firstly, of tools to design and support DQ-aware business processes (cf. workflow management system) and, secondly, of fault-tolerant HMIs. Still, the awareness problem must primarily be tackled by the management of a company.

Although bearing interdependencies, each of the eWBK’s input data is developed, on the one hand, during different stages of the PDP and, on the other hand, by various business units. Moreover, the synchronization routines among the IT systems and the information flows between the involved departments are not specifically tailored w.r.t. the claim for increased DQ of the new eWBK system. In the course of this thesis, a technique has been developed to detect inconsistencies—this technique is based upon a formal model of the input data which will be presented in the next chapter.
4 Formal Model of the Data

The content on the screens of the eWBK is compiled from three different data sources: the customer orders—derived from the CD, the BOM, and the AP (see Figure 1.2). All of the information is, firstly, produced and maintained by different business units and, secondly, subject to numerous changes during the product life cycle. In order to overcome consistency problems, a verification method has been developed in the course of this research project. Therefore, in this chapter a formal model of the input data including a collection of consistency requirements will be introduced.

In its first section, some fundamental concepts will be briefly explained. The subsequent sections will provide a formal model for each of the input data’s components. Section 4.5 describes the mapping of the eWBK system. Based on that, the final section will introduce consistency requirements to ensure high DQ of the eWBK system.

4.1 Fundamentals

Generally speaking, the economic system imposes the pressure of competition upon the market participants. In order to remain competitive, companies strive to continuously improve their day-to-day business. The corresponding optimization tasks are often characterized by high complexity: Considering the optimal planning of routes of a logistics company’s vehicle fleet, the optimal layout of components on a hardware device, the verification of complex software products, or the minimization of the throughput time of an assembly line, just to name a few. Due to their similar structure, many of these problems can be modeled by using mathematical formulas and be expressed, for example, as a constraint satisfaction problem or a combinatorial optimization problem (cf. vehicle routing problem, traveling salesman problem, knapsack problem).

Example 4.1: Eight queens puzzle
In 1848, the Bavarian chess player Max Bezzel (1824-1871) published the eight queens puzzle. Let an $8 \times 8$ chessboard and 8 queens be given. The task is as follows: Place the queens on the board so that no two queens attack each other. Bezzel asked for the number of possible placements.

In 1850, the correct number has been officially announced for the first time: 92. And several years later, in 1874, the English mathematician James Whitbread Lee Glaisher proved that no further solutions exist\(^1\).

\(^1\)Using the SWI Prolog (6.6.6) interpreter [74] and a modern computer (Intel Core 2 Quad Q9550 processor at 2.83 gigahertz (GHz) with 8 gigabyte (1 GB = 1024 MB) RAM), the computation of the solutions takes 0.187 seconds.
Puzzles marked by a similar structure are the four color map theorem and Sudoku. Problems of this kind can be generalized and expressed by a formula which allows to model various problems of mathematics and computer science:

**Definition 4.1: Constraint satisfaction problem (CSP)**

A CSP consists of a triple \( \langle V, D, C \rangle \) where \( V := \{ v_1, \ldots, v_n \} \) is a set of variables, \( D := \{ D_1, \ldots, D_n \} \) their finite domains \( (v_i \in D_i) \), and \( C := \{ c_1, \ldots, c_l \} \) is a set of constraints. Each constraint \( c_i \in C \) is a tuple \( \langle t_j, R_j \rangle \) where \( t_j \subset V \) is subset of \( k \) variables and \( R_j \) is a \( k \)-ary relation of the corresponding domains. A solution to the problem is given by an assignment to all variables so that all constraints are satisfied.

This formal model bears several advantages including a common basis for the development, implementation, and analysis of solving algorithms. In this context, various methods to model and represent knowledge were developed, among them propositional calculus, i.e. a field of mathematical logic that examines whether propositions are true or false, or how propositions can be combined using connectives like ‘and’ or ‘or’.  

**Example 4.2: Proposition**

Examples of propositions are given by sentences such as

"If it rains, then the street is wet" or

"If ten is bigger than five and five is bigger then three, then ten is bigger than three."

In the course of the emerging information processing industry, scientists contributed diverse data structures and programs to efficiently store and process such formalized knowledge with the help of computers, like decision diagrams (DDs) or the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [17]. Techniques of this kind were, and still are, subject to intense research (see, for example, [24]).

**Definition 4.2: Algorithm**

*Algorithms are clearly stated procedures composed of a finite number of tasks to effectively solve a given problem.*

Given increasing computing power, the limits of feasibility w.r.t. the size of problems are expanding and, likewise, the efficiency of daily operations is steadily advancing—despite the growing complexity. Considering modern information processing devices, the binary numeral system—positional system with base two—as well as elementary logic gates\(^3\) have emerged as fundamental components for the internal representation of information within computers due to a straightforward implementation in digital electronics. Regarding the design of digital circuits or the verification of software systems, the *Boolean satisfiability problem* (SAT)—as a special case of CSP—is playing a particularly important role.

\(^2\)See [67] for an introduction to logic, or [64] for a comprehensive overview of formal models within the field of product configuration.

\(^3\)Realization of logical operations by combining one or two input signals to produce one output signal, for example OR \((a \lor b)\), AND \((a \land b)\), or NOT \((\overline{a})\).
### 4.1 Fundamentals

#### Definition 4.3: Boolean satisfiability problem

The task to determine whether there exists an assignment to all variables of a Boolean function $f : B^k \rightarrow B$, $B := \{0, 1\}$ so that $f$ evaluates to 1 or to prove that no such assignment exists is denoted by the Boolean satisfiability problem.

#### Example 4.3: Boolean function

Let three Boolean variables $a$, $b$, and $c$ as well as the Boolean function

$$f(a, b, c) = (a \lor b) \land c$$ (4.1)

be given. This function only evaluates to 1 if $a$ or $b = 1$ and $c = 1$. In all other cases, $f$ will evaluate to 0.

Functions of this kind can be represented by truth tables (see Table 4.1). Truth tables depict all possible combinations of the input variables’ assignments (also: interpretations).

---

**Table 4.1: Truth table of (4.1)**

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$f(a, b, c)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 4.1: Different representations of (4.1)**

(a) Decision tree

(b) Reduced ordered binary decision diagram
as well as the corresponding output of the function. Obviously, \( f(a, b, c) \) is satisfiable—a simple example of an unsatisfiable function is given by \( f(a) = a \land \lnot a \). However, the Boolean functions of real-world applications usually comprise more than three variables. The truth table of a function with 100 variables would include \( 2^{100} \approx 1.27 \times 10^{30} \) entries, far too many for explicit storage.

This is why alternative representations of a Boolean function’s behavior, like decision trees (Figure 4.1, a) or reduced ordered binary decision diagrams (ROBDDs) (Figure 4.1, b) have been developed [37, 14]. The ROBDD is obtained by merging redundant nodes and paths of the decision tree (i.e. ‘Reduced’)—allowing to very compactly represent the function’s behavior. ‘Ordered’ means that the variables appear in the same order in each path from the top to the bottom node(s) (cf. Definition 6.1). While the nodes depict variables (circles), the outgoing edges represent the assignment of values to this variable (BDDs: either 0 or 1). The function’s output is stored in terminal nodes (rectangles). In such representations, any path from the top to the bottom reflects an interpretation. Next to compact storage, this data structure allows to be efficiently handled and manipulated with the help of computer programs.

Figure 4.2: Overview of both the relations among the entities of the input data and the responsible business units
4.2. Configuration Data and Customer Orders

Before introducing the formal model, Figure 4.2 provides an overview of the input data including the relevant relations among the entities. The red line between workstation and assembly step, for example, is interpreted in the following way: a workstation is connected to (also: references) at least one assembly step—more colloquially: The operator(s) of each workstation will carry out at least one assembly step. In turn, an assembly step is connected to exactly one workstation—more colloquially: An assembly step is carried out only once, at the corresponding workstation. Next to the ER model of the eWBK system, this figure illustrates the business units responsible for the data’s maintenance.

Table 4.2: Properties defining a car

<table>
<thead>
<tr>
<th>Property</th>
<th>Attribute</th>
<th>Domain</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brazil</td>
<td>$C_B$</td>
<td>$D_C$ (Country)</td>
<td>$v_C$</td>
</tr>
<tr>
<td>Germany</td>
<td>$C_G$</td>
<td>$D_C$ (Gearbox)</td>
<td>$v_G$</td>
</tr>
<tr>
<td>Russia</td>
<td>$C_R$</td>
<td>$D_C$</td>
<td>$v_C$</td>
</tr>
<tr>
<td>Automatic 7-gear</td>
<td>$G_{AT}$</td>
<td>$D_G$ (Gearbox)</td>
<td>$v_G$</td>
</tr>
<tr>
<td>Manual 6-gear</td>
<td>$G_{M6}$</td>
<td>$D_G$</td>
<td>$v_G$</td>
</tr>
<tr>
<td>4-wheel</td>
<td>$M_4$</td>
<td>$D_M$ (Mode of drive)</td>
<td>$v_M$</td>
</tr>
<tr>
<td>Front</td>
<td>$M_F$</td>
<td>$D_M$</td>
<td>$v_M$</td>
</tr>
<tr>
<td>Rear</td>
<td>$M_R$</td>
<td>$D_M$</td>
<td>$v_M$</td>
</tr>
<tr>
<td>Leather</td>
<td>$I_L$</td>
<td>$D_I$ (Interior decor)</td>
<td>$v_I$</td>
</tr>
<tr>
<td>Normal</td>
<td>$I_N$</td>
<td>$D_I$</td>
<td>$v_I$</td>
</tr>
<tr>
<td>Sport</td>
<td>$I_S$</td>
<td>$D_I$</td>
<td>$v_I$</td>
</tr>
<tr>
<td>75 kW Turbodiesel</td>
<td>$E_{75}$</td>
<td>$D_E$ (Engine)</td>
<td>$v_E$</td>
</tr>
<tr>
<td>100 kW Otto</td>
<td>$E_{100}$</td>
<td>$D_E$</td>
<td>$v_E$</td>
</tr>
<tr>
<td>125 kW Otto</td>
<td>$E_{125}$</td>
<td>$D_E$</td>
<td>$v_E$</td>
</tr>
<tr>
<td>Digital radio</td>
<td>$R_{DR}$</td>
<td>$D_R$ (Receiving unit)</td>
<td>$v_R$</td>
</tr>
<tr>
<td>None</td>
<td>$R_{NO}$</td>
<td>$D_R$</td>
<td>$v_R$</td>
</tr>
<tr>
<td>TV &amp; Digital radio</td>
<td>$R_{TV}$</td>
<td>$D_R$</td>
<td>$v_R$</td>
</tr>
</tbody>
</table>

4.2 Configuration Data and Customer Orders

The CD is maintained by the R&D, Marketing, and Sales department and contains information on the available equipment features of the products. This information is the basis for the communication between customers and company. During the configuration process (more and more companies provide online configurators via web interface to support this procedure), customers first choose a certain product—for example Audi A6. Next, all desired equipment features are specified, for example the engine, the gearbox, or the decor. This individual selection is finally represented by the corresponding customer order which, in addition, holds information on the customer, the destination country—like Brazil, and the desired date of delivery.
4.2.1 Design Space

The basis for the configuration of products is given by a set of properties defining the car (cf. Table 4.2 and Figure 4.3). First, the terms attributes, domains, and variables are introduced as follows.

**Definition 4.4: Attributes, domains, and variables**

Let the finite, non-empty set of symbols

\[ A := \{ a_1, \ldots, a_n \} \]  

be referred to as the set of attributes. The partition\(^4\)

\[ D := \{ D_1, \ldots, D_m \} \]

of \( A \) is referred to as the set of domains. The set

\[ V := \{ v_1, \ldots, v_m \} \text{ with } v_i \in D_i \]

is denoted as the set of variables.

The set of attributes and domains is the basis for the characterization of configurations which reflect the customers’ individual selections of the cars’ properties. Since the color of a car can either be red, green, or blue—or a car is equipped either with a 75 kW Turbodiesel engine or with a 100 kW Otto engine, the attributes of one domain mutually exclude each other. Therefore, the design space, i.e. the set of all configurations, is composed as follows.

\(^4\)A partition \( P \) is a set of non-empty, non-overlapping subsets of a given set \( S \), which has to satisfy the following condition:

\[ \bigcup_{P_i \in P} P_i = S \]
4.2. Configuration Data and Customer Orders

Definition 4.5: Design space and configurations
For a given \( A \) and \( D \), we define

\[
S := D_1 \times \cdots \times D_m
\]

(4.5)
to be the design space, and each element \( w \in S \) is denoted as configuration (also: customer order). Thus, the total number of configurations is given by:

\[
|S| := \prod_{1 \leq i \leq m} |D_i|
\]

(4.6)

Let a tuple \( w = (a_1, \ldots, a_m) \in S \) be given. Then, the mapping function

\[
M : S \to \{ z \mid z = \{a_1, \ldots, a_m\} \land (a_1, \ldots, a_m) \in S \}
\]

(4.7)
yields a set containing all attributes of \( w \). This mapping function is required to apply set-operators like, for example, in (4.9) or (4.34).

Summing up, while configuring a product, customers assign one value of each domain \( (D_j) \) to the corresponding variable \( (v_j) \). The resulting configuration is also called customer order.

4.2.2 Feasible Design Space

Due to technical and legal restrictions or marketing-related strategies, certain combinations of equipment features can become infeasible: For example, not every gear box can be combined with every engine. This is expressed by constraints. Constraints are specified by using the if-then paradigm (also: material implication \( \rightarrow \) ) of propositional calculus.

Example 4.4: Constraint in the configuration data
“\textbf{If} customer selects 75 kW Turbodiesel engine, then Automatic 7-gear \textbf{cannot} be selected.”

More formally:

\[
(v_E = E_{75}) \rightarrow (v_G \neq G_{A7})
\]

(4.8)

Hence, the constraint marks all configurations \( \tilde{w} \in S \) with

\[
\{E_{75}, G_{A7}\} \subseteq M(\tilde{w})
\]

(4.9)
as infeasible—accordingly, customers cannot choose such configurations.

In the following, a constraint is regarded as a set of configurations:

Definition 4.6: Constraint
A constraint is an \( m \)-tuple \( c := (P_1, \ldots, P_m) \), \( P_i \subseteq D_i, P_i \neq \emptyset \) which induces the set:

\[
W(c) := P_1 \times \cdots \times P_m
\]

(4.10)

Therefore, a constraint imposes a relation among a subset of variables \( \hat{V} \subseteq V \). This set is referred to as the constraint’s scope:

\[
s(c) := \{ v_i \mid P_i \neq D_i, 1 \leq i \leq m \}
\]

(4.11)
Using the above definition allows to rewrite the constraint of Example 4.4 in the following way:

\[
W(c) = D_C \times \{ G_{AT} \} \times D_M \times D_I \times \{ E_{75} \} \times D_R
\]  

(4.12)

Accordingly, the constraint’s scope is given by:

\[
s(c) = \{ v_G, v_E \}
\]  

(4.13)

As from now on we denote the set of constraints within the CD by:

\[
C_{CD} := \{ c_1, \ldots, c_k \}
\]  

(4.14)

Considering the design space \( S \) and the set of constraints \( C_{CD} \), the feasible design space is derived as follows:

\textbf{Definition 4.7: Feasible design space, Configuration problem}

For a given \( A, D \) and \( C_{CD} \), the feasible design space \( F \) is given by:

\[
F := S \setminus \bigcup_{1 \leq i \leq k} W(c_i), \quad c_i \in C_{CD}
\]  

(4.15)

Each \( w \in F \) embodies a feasible combination of equipment features—i.e. selectable by customers, and will henceforth be denoted as valid configuration. Following Definition 4.1, from now on the triple \( \langle V, D, C_{CD} \rangle \) is called configuration problem.

In conclusion, the CD provides a character set to encode the individual selections of customers considering the properties of cars, i.e. configurations. For various reasons, certain combinations of equipment features are not valid. Such restrictions are expressed via constraints utilizing the concept of if-then—as a consequence, the set of all configurations is split into valid and invalid configurations. While configuring the product, customers can only select equipment features so that all constraints are satisfied (cf. Definition 4.1), thus only valid configurations will be inserted into the production program.

4.3 Bill of Materials

The BOM contains information on all parts which are necessary to build a certain product. It plays an essential role in producing industries because it is the common basis for the product development process (PDP, cf. Section 2.1) and for the production planning and control (PPC, cf. Section 2.3). The BOM, therefore, is used by different business units and various applications. Next to a UID, the BOM incorporates additional information for each part, like its denotation, the necessary amount, or the kind of acquisition (in-house or external). According to the requirements of specific applications, further information is conveyed by the BOM, for example, the parts’ time schedule w.r.t. the product’s life cycle (cf. Redevelopment, Section 2.1.2), the responsible design engineer, or its size and weight—necessary for the planning of the transport and warehousing. In general, BOMs might be distinguished w.r.t. the following two properties:
Table 4.3: List of parts within the bill of materials

<table>
<thead>
<tr>
<th>UID</th>
<th>Denotation</th>
<th>Amount</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 A</td>
<td>75 kW Diesel engine</td>
<td>1</td>
<td>$E_{75}$</td>
</tr>
<tr>
<td>1000 B</td>
<td>100 kW Otto engine</td>
<td>1</td>
<td>$E_{100}$</td>
</tr>
<tr>
<td>1000 C</td>
<td>125 kW Otto engine</td>
<td>1</td>
<td>$E_{125}$</td>
</tr>
<tr>
<td>1001</td>
<td>Chassis</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Bodywork</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Wheel</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Front door</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Rear door</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Tailgate</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Cockpit</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1008 A</td>
<td>Steering wheel, Leather</td>
<td>1</td>
<td>$I_{L}$</td>
</tr>
<tr>
<td>1008 B</td>
<td>Steering wheel, Normal</td>
<td>1</td>
<td>$I_{N}$</td>
</tr>
<tr>
<td>1008 C</td>
<td>Steering wheel, Sport</td>
<td>1</td>
<td>$I_{S}$</td>
</tr>
<tr>
<td>1009 A</td>
<td>Automatic gear selector, Leather</td>
<td>1</td>
<td>$I_{L} \land G_{A7}$</td>
</tr>
<tr>
<td>1009 B</td>
<td>Automatic gear selector, Normal</td>
<td>1</td>
<td>$I_{N} \land G_{A7}$</td>
</tr>
<tr>
<td>1009 C</td>
<td>Automatic gear selector, Sport</td>
<td>1</td>
<td>$I_{S} \land G_{A7}$</td>
</tr>
<tr>
<td>1010 A</td>
<td>Manual gear selector, Leather</td>
<td>1</td>
<td>$I_{L} \land G_{M6}$</td>
</tr>
</tbody>
</table>

- **Structure (‘Modular BOM’)**
  Structural information within the BOM reflects is-part-of relations among the parts. Therefore, next to individual parts, also intermediate goods (like a cockpit) are listed. The car is composed, for example, of an engine, the interior, a bodywork, a chassis, and the seats (see Figure 4.4). Each of these components, likewise, is composed of further components and so forth. It is important to realize that this structure is standardized and—to a certain degree—applied to the entire product range. Structural relationships are provided by the design engineers and allow, at least partially, to conclude the sequence of the parts’ assembly (cf. assembly steps referencing parts and their mapping to workstations in Section 4.4).

- **Variance (‘Configurable BOM’)**
  Information of this kind is required whenever a product appears in many variations (cf. mass customization). Using this information allows to summarize the parts for all variants within one single BOM. In order to achieve this, constraints are introduced allowing to control the parts’ necessity w.r.t. customer orders. The set of parts, accordingly, is split into parts required for the production of all variants and parts required for the production of certain variants only (cf. three different engines in Table 4.3).
4. Formal Model of the Data

4.3.1 Parts and Constraints

Since the BOM of the eWBK system contains information on all parts of all variants (cf. Variance in Section 4.3), constraints are introduced allowing to restrict the necessity of parts to subsets of configurations.

**Example 4.5: Constraint in the bill of materials**

“The part 1009 A (Automatic gear selector, Leather) is required if customer selects Automatic 7-gear and Leather”

More formally:

\[(v_G = G_{AT}) \land (v_I = I_L) \rightarrow 1009 \text{ A is required}\] (4.16)

**Definition 4.8: Bill of materials**

We denote the set of parts within the BOM by:

\[P := \{p_1, \ldots, p_p\}\] (4.17)

Furthermore, we denote a constraint specified to control the necessity of a part \(p\) by \(c_p\) (cf. Definition 4.6). The following equation allows to assess the demand of parts. Let a configuration \(w\) be given, a part \(p\) is required for \(w\)’s production iff:

\[w \in W(c_p)\] (4.18)

A part is required for the production of all configurations if no constraint is specified.

Similar to constraints within the CD (cf. Definition 4.6), the constraint of Example 4.5 can be rewritten in the following way:

\[W(c_p) := D_C \times \{G_{AT}\} \times D_M \times \{I_L\} \times D_E \times D_R\] (4.19)

4.3.2 Part Types and Composite Constraints

In order to overcome the high number of variants among certain parts, like gear selectors, subsets of parts can be summarized within part types (cf. Table 4.4). Each part type can be...
Table 4.4: Aggregation of parts within part types

<table>
<thead>
<tr>
<th>UID</th>
<th>Denotation</th>
<th>Referenced Parts</th>
<th>Composite Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Automatic gear selectors</td>
<td>1009 *</td>
<td>$G_A^7$</td>
</tr>
<tr>
<td>002</td>
<td>Manual gear selectors</td>
<td>1010 *</td>
<td>$G_M^6$</td>
</tr>
</tbody>
</table>

... ... ... ... ...

linked to a composite constraint to specify the set of configurations where exactly one of the referenced parts will be required for production (also: exclusive disjunction)—every car, for instance, needs one steering wheel or one gear selector. This information is maintained by the logistics department to ensure the correct acquisition of parts.

**Example 4.6: Part type, composite constraint and referenced parts**

“If customer selects Automatic 7-gear, then either 1009 A or 1009 B or 1009 C is required.”

More formally:

$$(v_G = G_A^7) \rightarrow (1009 A \lor 1009 B \lor 1009 C) \text{ is required} \quad (4.20)$$

**Definition 4.9: Composite constraint**

The inclusive disjunction of constraints (Definition 4.6) is denoted by composite constraints

$$\hat{c} := c_1 \lor \cdots \lor c_l \quad (4.21)$$

inducing the set

$$W(\hat{c}) := \bigcup_{1 \leq i \leq l} W(c_i) \quad (4.22)$$

By substituting the logical ‘or’ operator ($\lor$) by a union operator ($\cup$), composite constraints are considered sets of configurations—similar to ordinary constraints.

It is important to realize, that the composite constraints of part types are manually maintained by logisticians and not automatically derived from the BOM (see Figure 4.2)—in this case the composite constraint of part type AA01, for example, would read as follows:

$$(G_{M6} \land I_B) \lor (G_{M6} \land I_N) \lor (G_{M6} \land I_S)$$

**Definition 4.10: Part type**

We denote the set of part types by $T := \{t_1, \ldots, t_l\}$. Let a configuration $w$ and a part type $t \in T$ be given, $t$ is mandatory for $w$’s production iff:

$$w \in W(\hat{c}_t), \quad (4.23)$$

where $\hat{c}_t$ is the part type’s composite constraint. A part type is required for the production of all configurations, when no composite constraint is specified. Each part type $t_j \in T$ references a set of parts. The mapping function

$$P^T : T \rightarrow 2^P \quad (4.24)$$
yields this set:

\[ P^T(t_j) = \{ p_{j1}, \ldots, p_{jn} \} \subseteq P \] (4.25)

Furthermore, to express that exactly one of a part type’s referenced parts will be installed, the following rule is introduced:

\[ \forall t \in T : \exists! p \in P^T(t) \mid W(\hat{c}_t) \cap W(c_p) \neq \emptyset \] (4.26)

### 4.4 Assembly Plan

The AP is the common basis for the planning and optimization of the assembly process (cf. Section 2.3.1). In contrast to the BOM, containing all required parts, the AP specifies all manual assembly steps—i.e. executed by operators—which are necessary to assemble the product. Nevertheless, a connection between the AP and the BOM is established since assembly steps reference the parts needed for their execution (see Figure 4.2). It is important to realize that the composite constraints of the assembly steps are manually maintained by the industrial engineers. Besides the parts to mount, each assembly step bears information on its execution time and necessary auxiliary handling devices: like cordless screwdrivers or lifting gear. Coupled with that, the AP covers ergonomic aspects like the amount of overhead work, the working height, reach zones, or the positioning of grab containers. Furthermore, the AP describes all manual quality assurance methods—embodied by test steps. In order to improve the ergonomic design of workstations, the quality of the product, or the throughput time, assembly steps and their mapping to workstations are subject to continuous optimization measures until the EOP.

The assembly process is described based on both the structure of the product and the structure of the assembly line (see Figure 4.5):

- **Product structure (see Figure 4.4)**
  During the PDP, in the first planning phase of the assembly process, the arrangement of the assembly steps is oriented w.r.t. the structure of the product. Since this structure is standardized across the whole product range, sets of similar assembly steps, like ‘Mount front-wheel, left’, can be compared w.r.t. different products. By doing so, potential optimizations w.r.t. the execution time, the set-up time, or the ergonomic design of workstations can be systematically assessed.

- **Assembly line structure (see Figure 2.7)**
  In the second planning phase, the assembly steps are mapped to the workstations. This structure does not only reflect the layout of the assembly line but also involves the hierarchy within the assembly environment. From a quantitative point of view, usually there is a one-to-one mapping between operators and workstations. However, the execution of the assembly steps of certain workstations can require more than one operator—like ‘Mount rear bench seat’. Every workstation belongs to a group which,
in turn, is part of a manufacturing segment and so forth. Mapping assembly steps within the structure of the assembly line requires to bear the sequence of assembly in mind—i.e. the sequence of mounting the parts in the product.

It is important to realize that the arrangement of workstations and the mapping of assembly steps induces a sequence w.r.t. the assembly of the corresponding parts. Another sequence is induced by the parts’ inherent is-part-of relations—both sequences must not contradict each other.

**Definition 4.11: Assembly Plan**

We denote the set of assembly steps within the AP by:

$$E := \{e_1, \ldots, e_e\}$$  \hfill (4.27)

The execution of assembly steps might be controlled by composite constraints (cf. part types and Definition 4.9). Let a configuration $w$ and an assembly step $e \in E$ be given, then $e$ is executed while $w$’s assembly iff:

$$w \in W(\hat{e}_e)$$  \hfill (4.28)

where $\hat{e}_e$ is the assembly step’s composite constraint. An assembly step is executed during the assembly of all configurations, when no composite constraint is specified.
Each assembly step can be associated with the parts involved in its execution\(^5\) (see Figure 4.2). The mapping function

\[ P^E : E \rightarrow 2^P \]  

(4.29)
yields all parts referenced by an assembly step:

\[ P^E(e_j) = \{p_{j_1}, \ldots, p_{j_n}\} \subseteq P \]  

(4.30)

Table 4.5: Visualization properties within assembly steps to acquire the content of the eWBK-screens

<table>
<thead>
<tr>
<th>Visualization property</th>
<th>Content on the screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Corresponding attribute of the customer order</td>
</tr>
<tr>
<td>Part type</td>
<td>Corresponding part of the BOM</td>
</tr>
<tr>
<td>Customer order</td>
<td>UID of the customer order</td>
</tr>
<tr>
<td>Free text field</td>
<td>Corresponding text</td>
</tr>
</tbody>
</table>

Furthermore, in the course of the eWBK project, certain assembly steps were enriched by either one of the visualization properties of Table 4.5. Whenever an assembly step with such a property is determined to be executed (4.28), the screen at the workstation will display the corresponding content.

As shown before, assembly steps can be associated with diverse visualization properties, among others with a part type. The mapping function

\[ E^T : T \rightarrow 2^E \]  

(4.31)
is introduced to determine all assembly steps which reference a part type \( t_j \):

\[ E^T(t_j) = \{e_{j_1}, \ldots, e_{j_n}\} \subseteq E \]  

(4.32)

4.5 Mapping

Both the mapping of assembly steps to workstations and the visualization properties according to Table 4.5 constitute the basis to acquire the information for the screens at the workstations. For each customer order \( w \) within the production program, the eWBK system—supported by the acquisition system—computes the content on the screens according to the following procedure (see Figure 4.6):

1. Determine the set of parts required for \( w \)'s production according to (4.18) (Note that this process is performed by the acquisition system).

\(^5\)For example pick, prepare, mount, or unwrap a certain part.
2. Determine the assembly steps executed while $w$’s production by applying (4.28).

2.1. Determine the assembly steps bearing a visualization property.

2.2. Determine associated workstations.

3. Determine visualization property according to Table 4.5.

3.1. Map content to the screens at the workstations.

3.2. Show content if customer order $w$ enters the workstation.

As shown above, the constraints within the different components of the input data are the basis to extract the information on the screens of the eWBK system. Therefore, the next section introduces rules to detect inconsistencies: Let, for example, an assembly step with the visualization property “part type” be given. It is important to check that for each valid configuration either both the assembly step and the corresponding party type are determined to be necessary, or none.

### 4.6 Consistency Requirements

The process of determining whether or not a system works correctly is built upon the requirements which are to be met. In the course of this section, consistency requirements that must be fulfilled by the input data will be introduced (cf. [6, 7, 8]). These rules are based on the formal model and on fundamental set operations including $\text{SetUnion}$, $\text{SetIntersect}$, and $\text{SetMinus}$ (see Figure 4.7).
4.6.1 Consistency of the Configuration Data

Whenever the constraints of the configuration data mutually exclude each other, the feasible design space $F$ (i.e. the set of valid configurations) will be empty. Thus, the constraints $c_i \in C_{CD}$ are said to be inconsistent iff:

$$F = \emptyset$$  \hspace{1cm} (4.33)

In order to guarantee that all attributes of all domains are selectable by customers, the following rule is introduced:

$$\forall a \in A : \exists w \in F \mid a \in M(w)$$  \hspace{1cm} (4.34)

This rule ensures that every attribute $a_i \in A$ is element of at least one valid configuration.

Next to the above requirements, each customer order $w$ needs to satisfy all constraints $c_i \in C_{CD}$. This requirement is expressed by the following proposition:

$$\{w\} \cap F \neq \emptyset$$  \hspace{1cm} (4.35)

4.6.2 Consistency of the Bill of Materials

'Slow-moving items' are parts that will never be used during the production process of any valid configuration. Therefore, a part $p \in P$ is said to be a slow-moving item if it infringes the following proposition:

$$W(c_p) \cap F \neq \emptyset$$  \hspace{1cm} (4.36)

This is illustrated in Figure 4.7 where the set $W(c_{p_3}) \cap F \neq \emptyset$ (red lines) and the set $W(c_{p_1}) \cap F = \emptyset$. Therefore part $p_1$ is determined to be a slow-moving item.

A part type $t \in T$ is said to be eligible if there is at least one valid configuration where it is determined to be necessary for production. This is expressed by the following term:

$$\forall t \in T : W(\hat{c}_t) \cap F \neq \emptyset$$  \hspace{1cm} (4.37)

The next rules are introduced to ensure that each valid configuration that is contained in the set $W(\hat{c}_t)$ will be equipped with exactly one part of the corresponding part type $t$. The first proposition ensures that there is no valid configuration for which a part type $t \in T$ is determined to be necessary and, at the same time, none of $t$’s referenced parts will be required:

$$\forall t \in T : \left[ W(\hat{c}_t) \setminus \bigcup_{p \in PT(t)} W(c_p) \right] \cap F \ni \emptyset$$  \hspace{1cm} (4.38)

In addition to that, for each part type the sets of configurations induced by the parts’ constraints need to be pairwise disjoint:

$$\forall p_k, p_l \in PT(t), k \neq l : [W(c_{p_k}) \cap W(c_{p_l})] \cap F = \emptyset$$  \hspace{1cm} (4.39)

This equation ensures that there is no valid configuration where two parts of the same part type will be assembled.
4.6.3 Consistency of the Assembly Plan

Assembly steps that are never executed w.r.t. any valid configuration will infringe the following proposition:

\[ \forall e \in E : W(\hat{c}_e) \cap F \neq \emptyset \]  

(4.40)

In order to ensure consistency regarding the connection between assembly steps and part types, the next proposition must hold for all part types \( t \in T \):

\[ \forall e \in E^T(t) : [W(\hat{c}_e) \triangle W(\hat{c}_t)] \cap F = \emptyset \]  

(4.41)

Equation (4.41) contains the symmetric difference of two sets\(^6\) and can be rewritten by the following two formulas:

\[ [W(\hat{c}_e) \setminus W(\hat{c}_t)] \cap F = \emptyset \]  

(4.42)

\[ [W(\hat{c}_t) \setminus W(\hat{c}_e)] \cap F = \emptyset \]  

(4.43)

In this way, it is ensured that there is no valid configuration which activates the visualization of an assembly step while the corresponding part type is not determined to be necessary, and vice versa.

Inconsistency among assembly steps and the referenced parts will infringe the following proposition:

\[ \forall e \in E : \left[ W(\hat{c}_e) \setminus \bigcup_{p \in PE(e)} W(c_p) \right] \cap F \neq \emptyset \]  

(4.44)

The above formula ensures that whenever an assembly step is executed, at least one of the referenced parts is determined to be necessary.

4.7 Summary

In this chapter, the formal model of the input data has been introduced. The CD serves as interface between customers and company allowing to specify an individual choice of the

---

\(^6\) \( A \triangle B = (A \setminus B) \cup (B \setminus A) \).
products’ properties. For example, each configuration bears information of the product’s color, its engine, or the interior decor. This has been modeled with the help of a finite set of attributes and a partition grouping attributes of the same kind: the domains (e.g. all available engines). Accordingly, the cross-product of the domains yields the set of all possible attribute combinations: the design space. In order to reflect technical or legal restrictions, constraints are introduced controlling the feasibility of such combinations—they are modeled using subsets of the domains. The attributes, domains, and constraints of the CD allow to derive the feasible design space containing all configurations that can be selected by customers.

Similar to the CD, both the BOM and the AP incorporate constraints to control the necessity of parts and assembly steps, respectively. These constraints allow to determine the set of parts and assembly steps required for a given customer order. During the assembly process, this information is combined to derive the information displayed on the screens of the eWBK system.

In order to detect inconsistencies among the input data, consistency requirements have been designed. They are based on the formal model and on simple set operations like $\text{SetIntersect}$, $\text{SetMinus}$, or $\text{SetUnion}$. However, the proposed requirements could be extended by exploiting further structural properties and dependencies within the input data.
5 Verification

More and more computer-controlled devices, like smartphones, PCs, washing machines, or home entertainment systems, are becoming part of everyday life. A similar development can be observed considering the number of computer-based components within safety-critical systems, such as cars, airplanes, power plants, or medical equipment. However, not only the integration of computers in daily routine but also the complexity, and thus the error-proneness of such systems has been increasing. This trend is aggravated by the pressure to reduce the TTM limiting the time to validate the freedom from defects. Although a malfunctioning device, for example a TV, will trouble its owner and probably cause additional costs for the manufacturer (cf. recall campaign), errors within safety-critical systems induce a threat to life or physical condition—hence, they are unacceptable. This is why methods and tools to ensure the correct operation of systems are becoming increasingly important. There exist several approaches to validate systems: fault tree analysis (FTA), failure mode and effects analysis (FMEA), testing, simulation, or formal verification\(^1\). While testing requires a realization of the system, for example in terms of a software prototype, formal verification is based on formal models (i.e. expressed by means of mathematical or logical formulas) unambiguously describing both the functionality of the system and the system’s desired behavior. Given both specifications, the system’s adherence to the requirements is then proved—this process can be automated and carried out by tools, like SAT-Solvers (cf. Definition 4.3, Section 5.2) or model checkers. In this chapter, a verification technique based on MDDs will be introduced. This data structure is specifically tailored w.r.t. the multiple-valued domains of the variables of the input data’s formal model (cf. Definition 4.4) and allows to implement set operations for an efficient verification of the consistency requirements.

After an introduction of fundamentals in Section 5.1.1, the most important programming paradigms of the MDD software are presented in Section 5.1.2. Then, in Section 5.1.3, the construction of the feasible design space and the verification process are explained. Finally, Section 5.2 introduces two models for a SAT-based technique to validate the results of the MDD-based technique.

5.1 MDD-based Technique

5.1.1 Multi-valued Decision Diagrams

In [66], the authors introduce a graph-based structure—similar to BDDs—for the representation and manipulation of discrete functions which they define as multi-valued decision

\(^1\)A distinction between the terms validation and verification can be found, among others, in [2].
diagram (also: function graph). Decision diagrams, either binary or multi-valued, are used in different fields of application, for example, to support the development of multiple-valued field programmable logic gates [43, 44]. In performance evaluation of communication systems, the state space of Markovian systems can often be compactly represented by MDDs [46, 16]. BDD- or MDD-based techniques have also been applied within the field of dependability and reliability analysis of fault-tolerant systems, for example to efficiently represent the state space of such systems, or to incorporate fault probabilities in terms of edge-values [75, 47, 42]. Another application utilizing decision diagrams is given by interactive product configuration [27, 50, 26]. Most generally, decision diagrams are considered data structures to efficiently encode large sets. Therefore, they are supported by a rich body of research and used in many further applications.

In contrast to BDDs, representing Boolean functions (cf. Example 4.3), MDDs are used to encode multiple-valued (also: discrete) functions:

**Definition 5.1: Multiple-valued function**

A function

\[ f : X_1 \times \cdots \times X_k \rightarrow Y \]  

(5.1)

is called multiple-valued, if the domain of each variable \(X_1, \ldots, X_k\) and the image of the function \(I \subseteq Y\) are finite sets. The image of a function is a subset of its co-domain and contains the function’s output for a given input.

**Example 5.1: Least common multiple**

In number theory, the least common multiple (LCM) of two integers is the smallest natural number that is divisible by both numbers without remainder. Let the function

\[ LCM : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{N} \]  

(5.2)

and two variables \(X_1 := \{1, 2, 3, 4\}\) and \(X_2 := \{1, 2, 3\}\) be given.

Figure 5.1 shows two MDDs representing Example 5.1. MDDs are directed acyclic graphs (DAGs), where non-terminal nodes\(^2\) represent the variables \(X_1, X_2\), and the function’s output \(Y\). The terminal nodes in (a) represent whether or not \((X_1, X_2, Y) \in R\) (also: indicator function), where \(R\) is the relation induced by the \(LCM\) criterion. For example \((2, 1, 1) \notin R\) because: \(\frac{1}{2} \notin \mathbb{N}\).

In this form of layout, edges always direct from upper to lower nodes (also: child nodes), hence all edges are represented as non-oriented lines. The maximum number of a node’s outgoing edges (also: branches) and the maximum number of child nodes depends on the cardinality of the represented domain. Each edge that is connecting a node with a child node embodies the assignment of the according value (this can be considered a decision). MDDs are called ordered when the variables appear in the same order in all paths from the top node to the bottom node(s) (cf. Definition 6.1).

---

\(^2\)Here, in contrast to the diagrams in Figure 4.1, all nodes are rectangular.
5.1. MDD-based Technique

In (b), all paths pointing to terminal nodes 0 are omitted, i.e. all elements of $X_1 \times X_2 \times Y$ that do not fulfill the LCM criterion. In addition to that, all equivalent sub-trees are summarized—see the gray nodes in (a) and (b). When the MDD is ordered and there are no more equivalent sub-trees, then it is called a reduced ordered multi-valued decision diagram (ROMDD)—for simplicity reasons henceforth just MDD. Finally, terminal nodes 1 are also omitted—the diagram shown in (b), therefore, implicitly represents all elements $X_1 \times X_2 \times Y \in R$. By doing so, the number of nodes and thus the memory consumption is reduced.

In the following, this approach is carried over to the field of product configuration: MDDs encoding sets of configurations (see Figure 5.2). Each path within these diagrams represents an element of the corresponding set. The red branches within (a), for example, yield the configuration

$$w = (C_G, G_{M6}, M_R, I_N, E_{100}, R_{TV}) \in S.$$  

Depending on the considered set, like the design space $S$ or a constraint $c$, nodes represent the domains $D_i$ or the subsets $P_i$. Similar to Figure 5.1 (b), terminal nodes are omitted.
By removing terminal nodes, the relation whether or not a configuration is feasible is no more reflected by the diagram. This knowledge must be maintained by the programmer in accordance with the formal model. Next to compact storage of large sets (real-world configuration problems contain $\approx 170$ variables, cf. Tables 7.1 and 7.8), the MDD structure is suited to carry out set operations in an efficient manner. This is enabled by diverse programming paradigms which will be introduced in the following section.

### 5.1.2 Programming Paradigms

#### 5.1.2.1 Nodes and Branches

According to the formal model, the cardinality of domains, i.e. the number of enclosed attributes, varies from one to another (cf. Table 4.2). Furthermore, while applying operations (cf. `SetMinus` in Figure 5.6 and the construction of the feasible design space in Section 5.1.3)
or during the application of dynamic variable ordering techniques (cf. Section 6.1.2), the number of branches per node can change as well. Consequently, nodes must maintain their branches in a dynamic manner; this has been achieved by implementing a linked list.

As shown in Figure 5.3, nodes point to a branch which, in turn, points to the next branch, and so forth (red arrows). Branches, except for the lowest level branches, point to a child node (black arrows). In addition to that, each branch stores the corresponding attribute’s value (Val), such as $C_B$. The number of branches pointing to a node is stored within the Refs variable—to quickly identify unused nodes. Unused nodes regularly occur while executing operations, and can be deleted to reduce the memory consumption (also: garbage collection).

Generally speaking, each node embodies an MDD. The node representing the first level is called root node. Considering diagrams including all levels—in accordance with the number of domains (or variables) of the respective configuration problem—nodes of lower levels symbolize sub-trees.

### 5.1.2.2 Hashing and Caching

In computer science, caches refer to fast buffers (i.e. either soft- or hardware components) helping to reduce the number of accessing slow media (like harddrives) or calculating expensive functions (like traversing all paths within a decision diagram). The basic procedure is as follows: Whenever data is requested or a certain function is invoked, at first the cache will be browsed for the corresponding result. In case of software caches, this operation is accelerated by using hash functions. If the result is contained (also: cache hit), it will be immediately passed on to the requesting unit without any delay. Otherwise (also: cache miss), the result is retrieved—for example from the hard drive or by invoking the function—passed on to the requesting unit and stored in the cache for future requests.

The MDD software implements two kinds of caches: the node-cache and the operations-cache (also: ‘Unique-Table’ and ‘Computed-Table’, cf. [13, 52, 65] or [51]). The node-cache
Before adding a node, it has to be determined whether or not it already exists. Since this important operation is carried out very often, without caching, the diagram would have to be traversed by recursive descent every time. In general, the node cache is implemented based on a hash table and linked lists (also: collision list)—see Figure 5.4 (a). For a given node $X$, first a hash function is invoked to determine its hash value and its position within the hash table (varying from 0 to $n - 1$):

---

3 Note that during the construction of the feasible design space (cf. Section 5.1.3), many temporal decision diagrams are generated. The node-cache allows diagrams to share nodes and to reduce memory consumption.

4 The MDD software uses recursion to traverse all paths within diagrams. This basic principle (cf. ‘Apply’ in [51]) is adapted in all operations, such as SetMinus (see Figure 5.6, Algorithm Z.1).
5.1. MDD-based Technique

\text{position} = \text{Hash}_1(X) \% n

where $n$ is the size of the hash table (usually a prime number). The hash function applied to other nodes might yield the same result, therefore a collision list is used to store nodes with colliding hash values. In such cases, the corresponding collision list has to be traversed. In order to accelerate this traversal, a second hash function is used:

$$H_2 = \text{Hash}_2(X)$$

Whenever a node is added to the cache (and diagram, respectively), this function is invoked and its output stored in the node. Based on these values, using Hash$_2(X)$ obviates the need for comparing all nodes and branches while traversing the collision list. Only if the values of the second hash function coincide, both nodes—including all of their branches—are compared.

In order to increase the performance of operations on MDDs, a second kind of cache has been implemented: the operations-cache. Since operations can differ w.r.t. their input and output (also: type signature), the MDD software implements one cache for each kind of operation. For example, binary operations\textsuperscript{5}, such as \textit{SetMinus} or \textit{SetIntersect}, require two input nodes and will yield one output node. For this, the software implements a special caching element which is stored in the hash table or collision list, respectively. These elements store pointers to both the two operands ($Op_1$ and $Op_2$) and the node representing the result ($Res$). Whenever a binary operation is invoked, at first the cache will be browsed for the result. This operation is accelerated by using a hash function (cf. node cache) applied to the two input nodes $Op_1$ and $Op_2$:

\text{position} = \text{Hash}(Op_1, Op_2) \% n

If more then one caching element is stored w.r.t. the hash value, the collision list has to be traversed. When the two input nodes coincide with the nodes of a caching element, the corresponding result is immediately returned. In the case of a cache miss, a new caching element with the two input nodes and the result will be created and inserted in the cache.

The performance of the MDD-based technique is heavily influenced by the implemented hash functions\textsuperscript{6}. Ideally, the chosen hash function distributes the hash values evenly over the positions of the hash table. This distribution and the length of the collision lists have been monitored during diverse experiments. In the end, the implemented hash functions—based on bit shifting and an XOR operation—turned out to be of sufficient quality. Nevertheless, when the length of the collision lists exceeds a certain value, garbage collection is invoked removing unused nodes.

\textsuperscript{5}There are also unary operations, like \textit{NodeCount} or \textit{PathCount}.
\textsuperscript{6}See [63], chapter 16, for an introduction to hashing.
5.1.2.3 Swapping Two Adjacent Levels

The basic operation behind dynamic variable ordering techniques is swapping two adjacent levels within the decision diagram, henceforth denoted by \textit{LevelSwap}. Most basically, dynamic variable ordering is applied to reduce the number of nodes within the diagram (see Figure 5.5, or [23, 59], or Section 6.1.2).

Let an MDD and two levels \(i\) and \(i+1\) be given as shown in Figure 5.5. At first, an auxiliary structure (list) is initialized with elements containing the values of the branches of the levels \(i\) and \(i+1\), as well as pointers to the nodes of level \(i+2\)—cf. element with red border in (a). In the next step, the values of the levels \(i\) and \(i+1\) are exchanged in each element of the list, then the elements are sorted w.r.t to the values of the new level \(i\). Based on that, the list elements are traversed and for each group of similar values of the new level \(i\)—the gray boxes in Figure 5.5 (b)—a new branch is added pointing to the elements of the new level \(i+1\). These branches, likewise, point to the nodes of level \(i+2\). This process is carried out with respect to each node of the level \(i\) and \(i+1\), respectively. In addition, all pointers of level \(i-1\) as well as the involved nodes’ \textit{Refs} variables are updated as well.

It is important to realize that the number of nodes can also increase when swapping two levels. Algorithm Z.2 describes the swap procedure.

Figure 5.5: The \textit{LevelSwap} of two adjacent levels within an MDD
5.1. Construction and Verification Process

In order to validate the consistency requirements (cf. Section 4.6), an MDD representation of the feasible design space \( F \) is required. According to Definition 4.7 and equation \((4.15)\), the feasible design space is obtained by subtracting the union of all constraints \( c \in C_{CD} \) from the design space \( S \). This operation is equivalent to a successive subtraction of the constraints from \( S \). The sequence of subtracting the constraints is referred to as constraint compilation sequence. In order to accelerate the construction process, constraints can be processed as bundles, which means that before carrying out the actual subtraction, a unification operation is applied to a subset of constraints. The cardinality of this subset is controlled by the bundle parameter (cf. Algorithm Z.3).

First, an MDD representation of \( S \) is constructed—see Figure 5.2 (a)—expressed by:

\[
\text{MDD}(S)
\]

Each node represents a domain \( D_i \in D \) and contains all of the respective attributes \( a_{ik} \in D_i \).

Next, each constraint \( c \in C_{CD} \) is represented in a similar way, where the nodes represent the sets \( P_1, \ldots, P_m \)—see Figure 5.2 (b)—expressed by:

\[
\text{MDD}(c)
\]

Given the MDD representations of \( S \) and \( c_1, \ldots, c_k \), a \textit{SetMinus} operation is gradually applied to construct the feasible design space \( F \). For this, the variable ordering of the MDD representations must coincide. The corresponding intermediate MDD representations are denoted by:

\[
\text{MDD(int.)}
\]
During the successive subtraction of all \( c \in C_{CD} \), the constraints are verified w.r.t. consistency by checking whether the resulting intermediate MDD representations are empty after each subtraction step—cf. (4.33). Figure 5.7 shows the MDD representation of \( F \) w.r.t. to the domains of Table 4.2 and the constraints of Table 6.1.

After having successfully constructed an MDD representation of the feasible design space \( F \)—expressed by

\[
\text{MDD}(F),
\]

the verification of further consistency requirements is carried out.

For example, to validate whether all attributes are selectable—cf. (4.34)—for each attribute \( a_{ik} \in A \), an MDD representation of the following constraints is constructed:

\[
c_{ik} = (D_1, \ldots, D_{i-1}, a_{ik}, D_{i+1}, \ldots, D_m)
\]  

(5.3)

Next, for each of these MDDs, a SetIntersect operation is carried out, i.e.:

\[
\text{MDD}(F) \cap \text{MDD}(c_{ik})
\]  

(5.4)

If the resulting MDD is empty, the corresponding attribute \( a_{ik} \) cannot be selected by customers.

The detection of slow-moving items within the BOM—cf. (4.36)—is carried out in a similar way: First, all constraints are represented as MDD. Based on them, and on \( \text{MDD}(F) \), a SetIntersect operation is invoked, i.e.:

\[
\forall p \in P : \text{MDD}(F) \cap \text{MDD}(c_p)
\]  

(5.5)

When the resulting MDD is empty, the corresponding part will never be used during the production of any valid configuration.
In general, all consistency rules (cf. Section 4.6) are checked based on the three operations \textit{SetMinus}, \textit{SetIntersect}, and \textit{SetUnion}.

5.2 Model for the SAT-based Technique

In order to validate the results of the MDD-based technique, a further approach has been implemented. It is based on Boolean formulas describing the configuration problem and the constraints within the BOM. Given these, a Java-based SAT-Solver [36] is used to check the formulas' satisfiability (Definition 4.3).

While other implementations require the SAT problem to be stated in conjunctive normal form (CNF)—containing only conjunctions of disjunctions, i.e. \( \bigwedge_i \bigvee_j (\neg x_{ij}) \)—this SAT-Solver is able to handle all kinds of linear constraints with Boolean variables. Note that the introduced SAT model only comprises the verification whether or not the feasible design space is empty (4.33) and whether the BOM contains slow-moving items (4.36). Nevertheless, both approaches will be compared w.r.t. run-time in Chapter 7. In the following, a common sequential approach is presented. In order to improve the run-time, the approach is extended by elastic variables in the subsequent section.

5.2.1 Sequential Approach

The sequential approach generates a SAT problem for each constraint within the BOM and is based on the following variables:

\[
\forall a_i \in A: a_i \in B = \{0, 1\}, \tag{5.6}
\]

where the binary variable \( a_i \) defines whether the corresponding attribute is part of the configuration (\( a_i = 1 \)) or not (\( a_i = 0 \)). Furthermore, the following constraints are introduced:

\[
\forall D_i \in D: \sum_{a_{ik} \in D_i} a_{ik} = 1 \tag{5.7}
\]

Equation (5.7) defines that exactly one attribute of each domain belongs to the configuration. The next equation defines the constraints \( c_i \in C_{CD} \) so that the corresponding attributes are excluded from the search space:

\[
\forall c_i \in C_{CD}: \sum_{P_p \in \{P_p | P_p \neq D_j\}} \sum_{a \in P} a < |\{P_p | P_p \neq D_j\}| \tag{5.8}
\]

The equations (5.7) and (5.8) define a set of linear constraints \( C_{SAT} \) with \( |C_{SAT}| = |D| + |C_{CD}| \). Based on that, a SAT-Solver is used to determine whether \( C_{SAT} \) is satisfiable, i.e. whether the feasible design space is not empty (4.33).

In order to detect slow-moving items according to (4.36), each \( p \in P \) is tested in a sequential procedure. For each \( c_p \in P \) the following constraint is constructed and added to
$C_{\text{SAT}}$:

$$
\sum_{P \in \{ P_j | P_j \neq D_j \}} \sum_{a \in P} a \geq | \{ P_j | P_j \neq D_j \} | \cdot c_p
$$

(5.9)

The part $p$ is a slow-moving item if the extended $C_{\text{SAT}}$ is unsatisfiable. This procedure is described in Algorithm Z.4.

### 5.2.2 Incremental Approach

To improve the run-time, an incremental approach is introduced next. It is based on elastic variables switching the constraints $c_p \in P$ on or off, respectively:

$$
\forall c_p \in P : c_p \in B = \{0, 1\}
$$

(5.10)

These binary variables define whether the constraint is active ($c_p = 1$) or not ($c_p = 0$). In addition to (5.7) and (5.8), for each $c_p \in P$ the following constraint is constructed and added to $C_{\text{SAT}}$:

$$
\sum_{P \in \{ P_j | P_j \neq D_j \}} \sum_{a \in P} a \geq | \{ P_j | P_j \neq D_j \} | \cdot c_p
$$

(5.11)

Equation (5.11) reflects Equation (5.9), but the constraint is only active if variable $c_p = 1$.

Based on that, the incremental approach creates one SAT problem containing all linear constraints (5.7), (5.8) and (5.11), i.e. $C_{\text{SAT}}$. Besides, in each iteration the search space is restricted by adding a constraint of the following form to $C_{\text{SAT}}$:

$$
\sum_{c_p \in P} c_p \geq 1
$$

(5.12)

This equation ensures that at least one constraint of the BOM is active. Then, a feasible solution satisfying the extended $C_{\text{SAT}}$ is searched. For each $c_p = 1$ in the feasible solution, the corresponding constraint $c_p$ is removed from $P$. This procedure is continued until either $P$ is empty or the SAT-Solver does not find a feasible solution. In this case, all remaining constraints $c_p \in P$ are not consistent and the parts are slow-moving items. To reduce the number of iterations, it is important to find as many active constraints in $P$ as possible in each iteration. Therefore, the backtracking strategy of the SAT-Solver must be adapted as follows: For each constraint in $c_p \in P$, the priority of the corresponding variable $c_p$ must be high, and the decision phase must be 1. The incremental approach is described in Algorithm Z.5.

---

Backtracking is a tree-based, recursive computer program that can be applied to help finding a problems’ solutions, like the arrangement of the eight queens (Example 4.1). Given a set of partial solutions (i.e. placement of a subset of queens), the algorithm iteratively extends this set (add other queens), and checks whether the set can be completely filled by queens not attacking each other—if they do, the algorithm steps back to the last valid placement.
6 Performance Optimization

During the construction of the feasible design space, nodes and branches are added to the MDD (see Figures 5.6 and 5.7). This, in turn, increases the amount of required memory and the run-time of the construction and verification process. Considering inapplicable variable orderings, the size of the MDD can grow exponentially, and in the worst case the feasible design space cannot be constructed due to the limitation of main memory (see Figure 6.1, gray line).

In order to successfully apply the MDD-based verification technique, various optimization measures have been investigated. The scope of these methods can be classified into variable ordering and constraint compilation sequence. While variable ordering refers to the arrangement of levels within decision diagrams, the constraint compilation sequence refers to the sequence of subtracting constraints during the construction of the feasible design space (cf. Algorithm Z.3).

The first section of this chapter will deal with variable ordering. It is subdivided into static and dynamic methods. The second section will deal with the optimization of the constraint compilation sequence.
6.1 Variable Ordering

It is well-known that the variable ordering, i.e. the arrangement of levels within decision diagrams, has an impact on the number of nodes (see Figure 6.2). The problem of determining good variable orderings is covered by diverse scientific articles, for example in [50, 62, 20] or [3]. According to [12], finding an optimal ordering—minimizing the number of nodes—for ROBDDs is NP-complete.

In general, static methods are distinguished from dynamic methods. Static methods are applied without constructing the decision diagram. In contrast to that, dynamic methods are applied online, i.e. while the decision diagram is constructed—by doing so, knowledge of its structural properties, like the number of nodes and branches, is exploited which helps to choose a good level arrangement. This knowledge comes along with the disadvantage of computational costs which—compared to static approaches—are considerably higher.

In this thesis, the term variable ordering is used according to the following definition:

**Definition 6.1: Variable ordering**

Let a set of variables $V := \{v_i, \ldots, v_m\}$ be given. A variable ordering $O$ defines a strict total ordering on this set (i.e. a binary relation using ‘$<$’ as infix that is transitive, antisymmetric, and total):

$$v_{i_1} < \cdots < v_{i_m}, \text{ } v_{i_k} \in V, 1 \leq i_k \leq m$$

For a given number of variables $m$, the total number of variable orderings (also: permutations) is given by the factorial $m!$—obviously, it is impossible to investigate all different variable orderings in reasonable time. Let 170 variables be given (cf. the dimensions of real-world configuration problems in Table 7.1), then the total number of permutations is $\approx 7.25 \times 10^{306}$.

---

1Since ROBDDs are a special case of ROMDDs, this proposition also holds for MDDs.
The term variable ordering also refers to the arrangement of levels within decision diagrams. Let a variable ordering $O$ be given, then a decision diagram is called ordered w.r.t. $O$ if the variables appear in this order in every path from the top to the bottom node(s).

6.1.1 Static Variable Ordering

Static variable ordering methods establish a variable ordering prior to the actual construction of the decision diagram. Therefore, these approaches rely on structural properties of the corresponding problem domain. Structural properties within configuration problems refer to the relations among variables which are induced by the constraints. In other domains, such as hardware design, structural properties refer to the interconnection of the input and output signals, for example of logic gates. Most generally, static methods focus on placing levels of strongly related variables next to each other. By doing so, later, during the construction of the feasible design space, the branching will be restricted to a small range of the diagram’s levels. The authors of [57] provide a comprehensive survey of static methods.

However, in order to improve the variable ordering of the automotive configuration problems, the following methods are introduced: Cluster according to [50], Force according to [3] as well as MSR, which has been developed in the course of this research project [9]. In addition to the above methods, every configuration problem comes with an initial variable ordering induced by the alphabetical ordering of the internal IDs of the domains within the database (henceforth denoted by Original). The impact of the methods’ application to the automotive configuration problems will be discussed later in Section 7.3.

6.1.1.1 The Cluster Method

According to [50], clusters within the constraint graph of a configuration problem can give hints for a good variable ordering. Such graphs reflect relations among the variables of a configuration problem and are defined as follows:

**Definition 6.2: Constraint graph**
The constraint graph of a configuration problem is a tuple $G := (N, E)$, where $N$ is the set of nodes and $E \subseteq N \times N$ is the set of weighted edges. For a configuration problem $\langle V, D, C_{CD} \rangle$, first every variable $v \in V$ is mapped bijectively to a node $n \in N$. Then, for each constraint $c \in C_{CD}$ and for each pair of variables $v_j, v_k \in s(c)$, the two corresponding nodes $n_j, n_k \in N$ are connected via an edge with weight 1—if the edge already exists, its weight is increased by 1.

For a configuration problem and the corresponding constraint graph, the central variable is defined as follows:

**Definition 6.3: Central variable**
Let a constraint graph $G$ be given. Furthermore, let the function

$$adj : N \to E$$

(6.2)
be given as follows:

$$adj(n) := \{ e \in E \mid e = (n, n') \lor e = (n', n) \}$$ (6.3)

Based on that, the central variable of a constraint graph $G$ is the node $n_c \in N$ with

$$\forall n \in N : \sum_{\text{weight}} adj(n) \leq \sum_{\text{weight}} adj(n_c)$$ (6.4)

The central variable, therefore, is the variable that is most strongly connected to other variables through the set of constraints. Figures 6.3 and 6.5 show the constraint graphs of a medium-sized and a large real-world configuration problem.

Clusters within the configuration problem’s constraint graph reflect variables that are strongly related to each other via the constraints. The idea of the Cluster method is to derive a variable ordering according to the clustering of the constraint graph: i.e. placing the variables within a cluster next to each other in the diagram. The method does not only suggest how to arrange the variables within a given cluster but also proposes an ordering of the clusters themselves.

Following this proposal, the clustering within the constraint graphs of the automotive configuration problems has been investigated by applying the Markov Cluster Algorithm (MCL) [18, 19] with an inflation parameter of five—the result is shown in Figure 6.4. Every
configuration problem P1 - P6 is characterized by one big cluster—containing 39% (P3) up to 92% (P5) of all variables (Cluster 1). While P1 (13), P3 (18), and P6 (13) are marked by a more fine-grained clustering, P4 (6) and P5 (5) are weakly clustered.

Figure 6.4: Clustering within the constraint graphs of the automotive configuration problems. While the x-axis contains the IDs of the clusters, the y-axis depicts the number of contained variables. The legend specifies the percentage of variables contained in the largest cluster.
6.1.1.2 The Force Method

The second static variable ordering method is called Force and was introduced in [3]. Similar to the Cluster approach, this technique aims on placing the levels of strongly connected variables next to each other in the diagram. According to (4.11), constraints impose relations among variables: i.e. the constraint’s scope. Therefore, each constraint is considered an hyper-edge connecting all variables in its scope. Based on random initial (1-dimensional) positions of the variables, each hyper-edge is marked by a center of gravity: i.e. the sum

Figure 6.5: Constraint graph of a real-world automotive configuration problem (P5) with $|V| = |D| = 160$ and $|C| = 3844$
of the variables’ positions divided by their number. At first, all relevant hyper-edges (or constraints) are determined for every variable. Accordingly, each variable is affected by a force considering the average of all of its hyper-edges’ centers of gravity. Then, after having moved the variables to the new positions, all centers of gravity are updated. This procedure is repeated until either a fix limit of iterations is reached, or the system is in a stable state, i.e. all variables have reached equilibrium state considering their positions and the centers of gravity.

In Figure 6.6, the result of applying the Force method to P5 is shown. Each line represents a variable, and its length depicts the number of constraints $c_j$ where $v_i \in s(c_j)$. The arrangement of lines—starting from the top to the bottom—corresponds to the variable ordering in the MDD (cf. Section 7.3). Obviously, the final positions of the variables strongly depends on the variables’ initial positions (randomly chosen). From now on, the initial positions of the variables were chosen in accordance with the Original ordering—and the resulting variable ordering is denoted by Force 1.

In order to overcome the non-deterministic behavior of Force and the weakly clustered structure within the constraint graphs of some configuration problems, a new method has been developed in the course of this project and will be introduced next.

6.1.1.3 The MSR Method

The MSR method was proposed in [9] and is inspired by algorithms known, among others, from the field of sensor and actor networks (SANETs)—considering, for example, fault-tolerant and robust localization [21]. It is based upon the idea of connecting mass points with linear-elastic springs. Hence, according to Hooke’s law, each mass point $m_i$ is affected
by the force

$$F_i := \sum_j F_{i,j} = \sum_j d(m_i, m_j) \cdot K_{i,j}. \tag{6.5}$$

The term $F_{i,j}$ denotes the force between mass point $m_i$ when being connected to $m_j$ via a spring. The term $d(m_i, m_j)$ depicts the distance between the two mass points. The scalar constant $K_{i,j}$ represents the spring constant for the spring connecting $m_i$ and $m_j$. Regarding SANET applications, mass-spring-relaxation algorithms aim at finding positions for the mass points so that the sum of the corresponding forces is minimized. In the following, this idea is applied to configuration problems for the purpose of identifying adequate static variable orderings for the MDD representations.

First, for each pair of variables $v_i, v_j$, a spring and the according spring constant $K_{i,j}$ are derived from the set of constraints $C_{CD}$:

$$n(i,j) := |\{c \in C_{CD}| \{v_i, v_j\} \subseteq s(c)\}| \tag{6.6}$$

$$K_{i,j} = K_{j,i} := \begin{cases} n(i,j) + |D_i| + |D_j| & \text{if } n(i,j) > 0 \\ 0 & \text{if } n(i,j) = 0 \end{cases} \tag{6.7}$$

In order to determine a distance between variables, all $v_i \in V$ are mapped to arbitrary positions $x_i$ in the interval $[0, 1] \in \mathbb{R}$. Then, for each variable the force $F_i$ is specified by the following formula:

$$F_i := \sum_j F_{i,j} = \sum_j (x_j - x_i) \cdot K_{i,j} \tag{6.8}$$

Given the vector $\mathbf{F}$ of the forces, the vector $\mathbf{X}$ of the positions, and the matrix $\mathbf{K}$ of the spring constants, the whole system can be written as a linear equation system (LES)—cf. (6.13):

$$\mathbf{K} \cdot \mathbf{X} = \mathbf{F} \tag{6.9}$$

Now, one variable $v_{i_0}$ is selected and mapped to the position 0, and another variable $v_{i_1}$ is selected and mapped to the position 1. The positions $x_{i_0}$ and $x_{i_1}$ are treated as constants whereas the positions of the remaining variables are regarded as unknowns.

According to the application within the SANET domain, the sum of forces affecting each variable $v_i \in V$ is minimized. In the variable ordering case, all $F_i$ with $i \neq i_0, i_1$ are set to zero. With these additional conditions, now the LES (6.9) can be solved. For a resulting $\mathbf{X}$, the following weighting function, which takes into account that $|F_{i,j}| = |F_{j,i}|$, is defined:

$$w(\mathbf{X}) := \sum_{i<j} |F_{i,j}| \tag{6.10}$$

The LES (6.9) is solved for each combination of variables $v_{i_0}, v_{i_1} \in V$. From the set of the resulting $\frac{m(m-1)}{2}$ solutions, we choose a solution $\mathbf{X}_{\text{opt}}$, given that $w(\mathbf{X}_{\text{opt}})$ is a minimum. For $\mathbf{X}_{\text{opt}}$ a variable ordering is obtained by transferring the natural ordering $(\leq)$ of the positions in $\mathbb{R}$ to the set of variables $V$ (cf. Definition 6.1).
The complexity of the algorithm is given by $O(m^5)$: Number of combinations $O(m^2)$ (cf. Algorithm 2.6, line 9 and 10) and the Gauss-Seidel method applied to an $m \times m$ matrix: $O(m^3)$ (cf. Algorithm 2.6, line 13). This complexity can be reduced further, for example by reusing the solution $X$ of the preceding step, or by restricting the number of combinations—considering the mapping of variables to the positions 0 and 1. In Table 7.2, the run-times of the MSR algorithm applied to the six automotive configuration problems are specified.

It is important to realize that if the constraint graph is not connected, certain variable mappings $v_i_0$ and $v_i_1$ will cause all other variables to accumulate at position 0 and 1, respectively. In such cases, the MSR method has to be applied within each component of the constraint graph. Furthermore, due to the symmetry of the problem, each variable ordering can be flipped horizontally—because this will not change the forces affecting each mass point. In order to overcome this ambiguity, the ordering is chosen according to the recommendations among others in [50, 15, 45]: Placing the most influential (i.e. central) variable in the upper region of the diagram (see Figure 7.9).

**Example 6.1: Mass-Spring-Relaxation**

Let the set of domains be given according to Table 4.2 and the set of constraints be given according to Table 6.1—see Figure 6.7 a).

---

2Note that the complexity of the Gaussian elimination has been used to estimate the complexity. The actual complexity of the Gauss-Seidel method is lower—and influenced by diverse factors: the residuum, the number of iterations, or the filling degree of the sparse matrix.
Table 6.1: Constraints for the MSR example

<table>
<thead>
<tr>
<th></th>
<th>$W(c)$</th>
<th>Spring constant</th>
</tr>
</thead>
<tbody>
<tr>
<td>$c_1$</td>
<td>$D_C \times {G_{A_r}} \times D_M \times D_I \times {E_{75}} \times D_R$</td>
<td>$K_{G,E} = 1 + 2 + 3 = 6$</td>
</tr>
<tr>
<td>$c_2$</td>
<td>$D_C \times D_G \times D_M \times {I_N} \times D_E \times {R_{TV}}$</td>
<td>$K_{I,R} = 1 + 3 + 3 = 7$</td>
</tr>
<tr>
<td>$c_3$</td>
<td>$D_C \times D_G \times D_M \times {I_L, I_N} \times {E_{125}} \times D_R$</td>
<td>$K_{I,E} = 1 + 3 + 3 = 7$</td>
</tr>
<tr>
<td>$c_4$</td>
<td>$(C_R) \times D_G \times {M_F, M_R} \times D_I \times D_E \times D_R$</td>
<td>$K_{C,M} = 1 + 3 + 3 = 7$</td>
</tr>
<tr>
<td>$c_5$</td>
<td>$D_C \times {G_{M_6}} \times {M_4} \times D_I \times D_E \times D_R$</td>
<td>$K_{G,M} = 1 + 2 + 3 = 6$</td>
</tr>
</tbody>
</table>

For each variable $v_i$, the force $F_i$ can be calculated as follows:

$$F_C = F_{C,G} + F_{C,M} + F_{C,I} + F_{C,E} + F_{C,R} = (x_G - x_C) \cdot K_{C,G} + \cdots + (x_R - x_C) \cdot K_{C,R}$$

$$F_R = F_{R,C} + F_{R,G} + F_{R,M} + F_{R,I} + F_{R,E} = (x_C - x_R) \cdot K_{R,C} + \cdots + (x_E - x_R) \cdot K_{R,E}$$

This set of equations can be transformed to

$$F_C = -(K_{C,G} + K_{C,M} + K_{C,I} + K_{C,E} + K_{C,R}) \cdot x_C + K_{C,G} \cdot x_G + \cdots + K_{C,R} \cdot x_R$$

$$F_R = -(K_{R,C} + K_{R,G} + K_{R,M} + K_{R,I} + K_{R,E}) \cdot x_R + K_{R,C} \cdot x_C + \cdots + K_{R,E} \cdot x_E$$

and finally be represented by the following LES:

$$\begin{bmatrix}
K_{G,C} & K_{C,G} & K_{C,M} & K_{C,I} & K_{C,E} & K_{C,R} \\
K_{G,M} & K_{M,C} & K_{M,G} & K_{M,I} & K_{M,E} & K_{M,R} \\
K_{I,G} & K_{I,M} & K_{I,C} & K_{I,E} & K_{I,R} & K_{I,*} \\
K_{E,G} & K_{E,M} & K_{E,I} & K_{E,C} & K_{E,E} & K_{E,R} \\
K_{R,G} & K_{R,M} & K_{R,I} & K_{R,E} & K_{R,R} & K_{R,*}
\end{bmatrix}
\begin{bmatrix}
x_G \\
x_I \\
x_R
\end{bmatrix} =
\begin{bmatrix}
F_G \\
F_M \\
F_I \\
F_E \\
F_R
\end{bmatrix}$$

According to (6.12), the diagonal elements $K_{i,*}$ are defined as follows:

$$K_{i,*} := -\left( \sum_{j=0}^{i-1} K_{i,j} + \sum_{j=i+1}^{m} K_{i,j} \right)$$

Note that the set of constraints and the corresponding spring constants remain constant throughout the MSR algorithm’s application. Now, $v_C$ and $v_R$ are mapped to the positions
0 and 1, the forces affecting the other variables are set to 0, and the spring constants are inserted into \( K \):

\[
\begin{pmatrix}
-7 & 0 & 7 & 0 & 0 & 0 \\
0 & -12 & 6 & 0 & 6 & 0 \\
7 & 6 & -13 & 0 & 0 & 0 \\
0 & 0 & 0 & -14 & 7 & 7 \\
0 & 6 & 0 & 7 & -13 & 0 \\
0 & 0 & 0 & 7 & 0 & -7 \\
\end{pmatrix}
\begin{pmatrix}
x_G \\
x_M \\
x_I \\
x_E \\
x_C \\
x_R \\
\end{pmatrix} =
\begin{pmatrix}
0 \\
x_G \\
x_M \\
x_I \\
x_E \\
F_C \\
\end{pmatrix}
\]

The first and last row of \( K \) can be omitted because the positions \( x_C \) and \( x_R \) are constant and the according forces \( F_C \) and \( F_R \) do not affect the solution. Furthermore, due to \( x_C = 0 \), the first column of \( K \) can be omitted, and since \( x_R = 1 \), the last column of \( K \) is transferred to the right side:

\[
\begin{pmatrix}
-12 & 6 & 0 & 6 \\
6 & -13 & 0 & 0 \\
0 & 0 & -14 & 7 \\
6 & 0 & 7 & -13 \\
\end{pmatrix}
\begin{pmatrix}
x_G \\
x_M \\
x_I \\
x_E \\
\end{pmatrix} =
\begin{pmatrix}
0 \\
x_G \\
x_M \\
x_I \\
-7 \\
0 \\
\end{pmatrix}
\]

The vector

\[
X =
\begin{pmatrix}
x_G \\
x_M \\
x_I \\
x_E \\
\end{pmatrix} =
\begin{pmatrix}
0.40625 \\
0.18750 \\
0.81250 \\
0.63500 \\
\end{pmatrix}
\]

solves this LES. After having determined the relaxed position of the variables, now the weighting function (6.10) is applied:

\[
w (X) = 6.5694
\]

The fixed variables and the vector of positions can be transferred into a variable ordering—see Figure 6.7 (b):

\[
O = v_C < v_M < v_G < v_E < v_I < v_R
\]

This calculation is carried out for each pair of variables—in the end, the optimal solution for this example is:

\[
O = v_C < v_M < v_G < v_R < v_I < v_E
\]

### 6.1.2 Dynamic Variable Ordering

In contrast to static methods, dynamic methods change the arrangement of the diagram’s levels during (or after) the construction. These methods, accordingly, exploit knowledge of the actual memory consumption which is determined by both the number of nodes and branches. In general, all dynamic variable ordering methods are based on swapping two adjacent levels in the decision diagram (cf. Section 5.1.2.3). In the following, three approaches
will be introduced—the impact of these methods upon the MDD representations of the automotive configuration problems is discussed in Section 7.4.

6.1.2.1 The Sifting Method

The Sifting method was invented by Rudell (cf. [59]). The idea behind this algorithm is to find an optimal position for one variable, given that the positions of all other variables remain constant. To achieve this, the variable (or level) under investigation is moved stepwise to the top and bottom of the diagram (by applying LevelSwap). Then, the arrangement of levels which requires the least memory consumption is chosen to be the new variable ordering. This approach is applied to all variables (or levels) within the diagram.

It is important to realize that during the application of the Sifting algorithm, the size of the decision diagram can increase. Therefore, the Sifting procedure was extended by a window approach allowing to restrict the range of moving the variable under investigation—expressed by

\[ SiftingX, \]

where \( X \) is the size of the window. This window is moved over the whole range of levels, i.e. from the top to the bottom and back again. For a window size \( s \) and a number of variables \( m \), the number of level swaps is given by:

\[ (s - 1) \cdot (2m - s) \tag{6.21} \]

Algorithm Z.7 describes the Sifting method.

6.1.2.2 The Permute Method

The Permute method is inspired by the articles of Fujita et al. [23] and Ishiura et al. [31], respectively. In contrast to Sifting, this method exhaustively investigates all permutations of level arrangements within a restricted window—allowing to obtain a local optimum. Let for example a window with a size of five be given, then all \( 5! = 120 \) different arrangements of variables within this window are evaluated.

The arrangement causing the least memory consumption is chosen to be the new variable ordering. Similar to Sifting, the window is moved over the whole range of variables, i.e. from the top to the bottom and back again—expressed by

\[ PermuteX, \]

where \( X \) is the size of the window. Note that due to the factorial growth, the window size should be kept small to reduce the computational effort. For a given window size \( s \) and a given number of variables \( m \), the total number of level swaps is given by:

\[ s! \cdot (m - (s - 1)) \cdot 2 \tag{6.22} \]

Algorithm Z.8 describes the Permute method.
6.1.2.3 The Partial Method

In order to reduce the computational effort of the Permute approach, the Partial algorithm was implemented. Similar to Sifting and Permute, Partial is parametrized with a window size—expressed by

\[ \text{Partial}_X, \]

where \( X \) is the size of the window. But in contrast to Permute, only a subset of all permutations within this window is evaluated; in turn, the window size can be extended, which causes less computational cost compared to Permute. For a given window size \( s \) and a given number of variables \( m \), the total number of level swaps is given by:

\[ 2^{(s-1)} \cdot (m - (s - 1)) \cdot 2 \quad (6.23) \]

Algorithm Z.8 describes the Permute method.

In general, the way of how to control the application of dynamic methods during the construction of the feasible design space is illustrated by Algorithm Z.11. Later, in Section 7.4, the impact of the different dynamic variable ordering approaches will be discussed.

6.2 Constraint Compilation Sequence

During the construction of the feasible design space \( F \), the sequence of compiling the constraints \( c \in C_{CD} \) determines the dynamics of memory consumption. Since the size of intermediate MDD representations can exceed the memory limitation—even if a good variable ordering is used—this sequence must be optimized to keep the decision diagram small.

In the following, three different approaches are introduced: NW according to [50], Sortlevel according to [6, 7], and Bestfit according to [9]. Similar to the static variable ordering (cf. Section 6.1.1), each configuration problem comes with an initial compilation sequence which is induced by the ordering of the constraints’ IDs within the database (henceforth denoted by Original). Section 7.5 discusses the application of these methods to the automotive configuration problems.

6.2.1 The NW Method

The NW method was proposed by Narodytska et. al. in [50] and is based on the assumption of having a strongly clustered structure within the constraint graphs of the configuration problems—similar to the proposed static variable ordering heuristic (cf. Section 6.1.1.1 and Figure 6.4). The basic idea of the NW method is to compile the constraints in a sequence so that, firstly, the number of affected variables is kept low in each step (i.e. when a constraint is compiled) and, secondly, the clustered structure of the configuration problems is kept in mind.
In the course of this research project, the NW method has been adopted to be carried out statically, i.e. before the actual MDD representation is constructed. The detailed algorithm can be found in [50].

6.2.2 The Sortlevel Method
In the course of this thesis, the dynamic Sortlevel approach was developed [6, 7]. The basic idea of this approach is to combine knowledge of the variable ordering along with the scopes of the constraints, and to compile the constraints in a sequence so that the branching after each step is restricted to a small range of the decision diagrams’ levels. For a given variable ordering $O$ and a constraint $c$, the following helper function is defined:

$$l(O, c) := \{i | v_i \in \text{scope}(c)\}$$

This function yields the positions of the variables within the ordering—in terms of natural numbers. Based on that, the following distance can be calculated:

$$d(c) := \max \{l(O, c)\} - \min \{l(O, c)\}$$

The distance, therefore, determines the maximal number of levels between the variables affected by a constraint. Now a partial order on the set of constraints is induced by sorting them in an ascending order according to their distances. This sequence, in turn, is used during the construction of the feasible design space.

In contrast to the NW method, this method is carried out dynamically, i.e. during the construction. The sorting of constraints is carried out every time the variable ordering is changed—for example due to the application of dynamic variable ordering methods—because hereby the distances of the constraints will change. The corresponding procedure is described by Algorithm Z.9.

6.2.3 The Bestfit Method
The Bestfit strategy, likewise, was developed in the course of this project [9] and—in contrast to NW and Sortlevel—is based on knowledge of the actual memory consumption of the decision diagram. The basic idea of this method is to compile the constraints in a sequence which keeps the memory consumption after each step minimal. Therefore, during the construction of the feasible design space, the best-fitting constraints are compiled first. For the method’s description, the following definitions are required:

$$\hat{C}_{CD} \subseteq C_{CD}, \text{ with } |\hat{C}_{CD}| = l$$

$$R = C_{CD} \setminus \hat{C}_{CD}$$

$$\text{MDD(int.)} = \text{MDD}(S) \setminus \bigcup_{c \in \hat{C}_{CD}} \text{MDD}(c)$$
In each step, every remaining constraint $c_r \in R$ is subtracted from MDD(int.), and while the memory consumption of MDD(int.)\MDD(c_r) is memorized, the resulting MDDs are discarded. Based on these findings, the best-fitting constraint $c_{bf}$, i.e. the one with the least memory consumption, is selected to be compiled next. By doing so, a flatter growth of the memory consumption is obtained—but computational costs increase.

This is why the method has been enhanced by a window approach so that in each step only a limited number of constraints $\tilde{R} \subset R$ is taken into account—expressed by

$$ \text{Bestfit}_X $$

where $X$ is the cardinality of $\tilde{R}$. The selection of this subset of constraints is shifted over $R$, and in each step $t$, the set $\tilde{R}_t$ is constructed so that all $|\tilde{R}_{t-1}|/2$ best-fitting uncompiled constraints of the preceding step $t-1$ become elements of the new $\tilde{R}_t$. Algorithm Z.10 describes the procedure.
7 Results

This chapter presents the results of the MDD-based verification technique applied to six highly customizable products—comprising 31 variants altogether—of the AUDI AG. It will be demonstrated that applying the proposed methods will allow to construct an MDD-based feasible design space representation in all cases.

The content is structured as follows: The first section will describe the toolchain, its components, and their interaction. Then, in Section 7.2, the real-world automotive configuration problems are introduced. In the subsequent section, the impact of static variable ordering methods will be presented. Even with good static variable orderings, the construction of the MDD representation of the feasible design space $F$ was not possible—considering all of the investigated products. Therefore, dynamic variable ordering methods had to be applied during the construction—cf. Section 7.4. Section 7.5 focuses on the methods’ potential to enhance the constraint compilation sequence. Finally, Section 7.6 summarizes the performance and run-times of the MDD-based and SAT-based approach.

7.1 Toolchain

The toolchain for the verification process consists of several components: the pre-processing software, the MDD software, and the two black-box applications MCL and SAT (see Figure 7.1).

The pre-processing component was written in C++ (consisting of approx. 5000 lines of code) and is responsible for diverse tasks, including the processing and transforming of the input data, the application of static variable ordering methods (cf. Section 6.1.1), and the application of the static constraint compilation sequence method $NW$ (cf. Section 6.2.1). Besides, this component provides an interface (via text file) for the MDD software, the SAT-Solver, and the MCL software.

The MDD software was written in C++ (consisting of approx. 10000 lines of code) and implements the programming paradigms introduced in Section 5.1.2. Next to structural elements, caches, and diverse operations—like $SetMinus$, this software implements both dynamic variable ordering methods (cf. Section 6.1.2) and methods to enhance the compilation sequence (i.e. $Sortlevel$ and $Bestfit$). During the construction of the feasible design space $F$, both dynamic ordering methods and methods to enhance the compilation sequence can be applied—their application (controlled in accordance with Algorithm Z.11) is parametrized within the interface file. Next to the parametrization, the file encodes a given configuration problem (in terms of the domains and constraints $c \in C_D$), as well as the constraints within the BOM and AP.
Once a good variable ordering for a configuration problem is found, it can be memorized and re-used whenever the feasible design space must be reconstructed—the same applies for the constraint compilation sequence—this information is also encoded within the interface file.

The black-box application MCL was downloaded from [19]. Similar to the MDD software, the input (i.e. the constraint graph) is generated by the pre-processing software. Based on that, the MCL application determines the clusters within the constraint graph and returns the information (via text file) to the pre-processing software, which, in turn, computes the corresponding static variable ordering (cf. Section 6.1.1.1).

The SAT-Solver was implemented using the SAT4J library [36]. For this, an interface to specify the configuration problem (cf. Section 5.2) was implemented using Java (consisting of approx. 1000 lines of code).

Before introducing the automotive configuration problems, it is necessary to briefly discuss the relation between the memory consumption and the number of nodes and branches. Considering BDDs, the number of nodes allows to directly infer the memory consumption because each node has two branches (given that each branch requires the same amount of memory). In contrast to that, MDD nodes are marked by different numbers of branches—therefore, the number of nodes alone is not sufficient when measuring the size (see Figure 7.2). This is why the MDD software does not only monitor the number of nodes but also the actual memory consumption. The following diagrams will only illustrate the actual size, i.e. the memory consumption in Megabyte (MB).
7.2. On the Configuration Problems

This section introduces all investigated real-world automotive products that were subject to the MDD-based verification technique. In Table 7.1, the cardinality of the corresponding configuration problems is shown. It contains six products P1–P6, each of them comprising one or more variants. Altogether, 31 variants of the whole product range (consisting of 53 variants, Figure 2.1, a) have been investigated.

While P3 is a product still under development, P4 and P5 are produced for a considerable long time, and in turn, their availability—considering the number of different markets—is the highest. The corresponding variable (i.e. Countries) is used to reflect numerous market-specific requirements. However, due to the fact that all products are delivered in many countries, this variable constitutes the central variable of each of the product’s constraint graph. In general, the number of domains (157–181) and attributes (781–1240) does not vary much among the different products. In contrast to that, the number of constraints \( C_{CD} \) varies from 2161 (P6) to 4679 (P3)—the same applies for the set of parts within the

![Figure 7.2: Dynamics of memory consumption and the number of nodes](image)

<table>
<thead>
<tr>
<th>Product</th>
<th>Variants</th>
<th>Attributes</th>
<th>Domains</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CD</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>989</td>
<td>168</td>
<td>2487</td>
</tr>
<tr>
<td>P2</td>
<td>7</td>
<td>1001</td>
<td>176</td>
<td>3040</td>
</tr>
<tr>
<td>P3</td>
<td>8</td>
<td>943</td>
<td>174</td>
<td>4679</td>
</tr>
<tr>
<td>P4</td>
<td>5</td>
<td>1240</td>
<td>181</td>
<td>4413</td>
</tr>
<tr>
<td>P5</td>
<td>6</td>
<td>1116</td>
<td>160</td>
<td>3844</td>
</tr>
<tr>
<td>P6</td>
<td>4</td>
<td>781</td>
<td>157</td>
<td>2161</td>
</tr>
</tbody>
</table>

7.2. On the Configuration Problems
BOM (P6: 13644 and P2: 25735). The BOM of products characterized by a higher number of variants contains more parts controlled by constraints. Note that only for P3 and P4 information of the assembly plan and the part types was available.

It turned out that the number of constraints $c \in C_{CD}$ alone does not allow to draw conclusions w.r.t. the memory consumption of the feasible design space representations (cf. P3 and P5 in Table 7.8). Therefore, the constraints scopes were investigated. Figure 7.3 depicts the number of constraints with a specific scope size (i.e. varying from two to eight because eight is the maximum size w.r.t. all configuration problems). Note that for the scope
On the Impact of Static Variable Ordering

Table 7.2: Performance of the static variable ordering methods

<table>
<thead>
<tr>
<th>Product</th>
<th>Run-time [s]</th>
<th>Original</th>
<th>Cluster</th>
<th>Force</th>
<th>MSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>0.001</td>
<td>0.146</td>
<td>31.063</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>0.001</td>
<td>0.089</td>
<td>47.600</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P3</td>
<td>0.001</td>
<td>0.128</td>
<td>37.961</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P4</td>
<td>0.001</td>
<td>0.614</td>
<td>44.725</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P5</td>
<td>0.001</td>
<td>0.262</td>
<td>44.873</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P6</td>
<td>0.001</td>
<td>0.079</td>
<td>25.774</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

sizes six, seven, and eight, a different scaling was used (red). The two smallest problems—from a memory consumption point of view, i.e. P3 and P6—have the least number of constraints with a scope size larger than four. Vice versa, the largest problem P5 is marked by numerous constraints with scope size > 4, and the percentage of binary constraints (i.e. scope size two) is relatively small: only 39%.

This allows to conclude that not only the pure number of constraints but also their complexity—for example in terms of their scopes’ sizes— influences the memory consumption of the MDD-based feasible design space representation. This observation is partially confirmed w.r.t. the clusters of the problems’ constraint graphs in Figure 6.4: the two low-consuming problems (P3 and P6) are marked by a more fine-grained clustering than the four larger problems.

7.3 On the Impact of Static Variable Ordering

This section presents the impact of the different static variable ordering methods (see Section 6.1.1). First, Table 7.2 displays the run-times of the methods. These computations were carried out by the pre-processing component using one core of an Intel Core 2 Quad Q9550 processor at 2.83 GHz with 8 gigabyte (GB) RAM. Compared to Cluster and Force, the MSR method requires considerably more time. Still, all methods compute the corresponding variable ordering in reasonable time.

In Figure 7.4 a), an overview of the resulting variable orderings of all static methods is given. Here (and also in Figures 7.9 and 7.10), each line represents a variable \( v_i \in V \). The length of each line depicts the number of constraints \( c_j \) where \( v_i \in s(c_j) \). The arrangement of the lines, from top to bottom, corresponds to the arrangement of the levels within the MDD. The central variable is marked by the red line, the red number corresponds to the number of constraints having the central variable in its scope. In (b), the symmetric \( m \times m \) adjacency matrices of the constraint graph are shown w.r.t. the corresponding variable orderings. The brighter a dot is, the more constraints induce a relation among the corresponding pair of variables.
In order to measure the performance of the different static approaches, the following experiments are carried out (using one core of an Intel Xeon E5-2643 processor at 3.3 GHz with 64 GB RAM). In the first experiment, the three static approaches and the Original arrangement are used as initial ordering. Based on that, the MDD software starts to compile the constraints using the Bestfit compilation sequence with a window size of 300 (Bestfit300).
In order to increase the initial orderings’ influence upon both the construction process and the size of \( \text{MDD}(F) \), the offset for the application of the dynamic method is set to 1000. Then, the prototype starts to dynamically re-order the variables using the \textit{Permute} method with a window size of five (\textit{Permute5}) and one maximal iteration. Re-ordering events are triggered by a size factor of three (cf. Algorithm Z.11). Based on this parameter set, the quality of the static methods can be compared in terms of the overall construction time, the size of \( \text{MDD}(F) \), and the number of performed re-orderings. Each of the above measures is depicted by the legends of Figure 7.5 showing the results of this experiment applied to P6.

Both orderings \textit{Original} and \textit{Force 1} cause considerably more memory consumption than \textit{Cluster} and \textit{MSR} during the compilation of the first 1000 constraints. The first application of \textit{Permute5} (after 1000 constraints have been compiled) greatly reduces the memory consumption in all cases. In addition, the higher the number of dynamic ordering events, the more time is required by the construction process. In this experiment, the \textit{MSR} approach produced the least memory consumption and the least number of dynamic ordering events.
7. Results

Figure 7.6: Comparison of the four different static variable orderings applied to the configuration problem P2. For this experiment dynamic reordering is disabled, therefore, the set of constraints to compile is restricted to 300.

For the second experiment, the set of constraints to compile is restricted to 300 binary constraints, and dynamic ordering methods are disabled. The quality of the static methods can be compared in terms of both the time for the construction process and the size of MDD($F$)—these measures are given in the legend of Figure 7.6, showing the experiment’s application to P2. Having started with the Original ordering, the prototype was not able to construct the feasible design space since MDD(int.) exceeded the memory limitation of 1024 MB.

Summing up, while the resulting variable ordering of Force depends on the randomly chosen, initial position of the variables (see Figure 6.6), Cluster clearly arranges the variables according to the clusters within the constraint graph. But Cluster relies on the external black box tool MCL and a strongly clustered structure within the constraint graph—which is not always the case (cf. P4 and P5 in Figure 6.4). The run-time of the static approaches in the pre-processing step is in all cases acceptable. However, since the MSR approach has a higher complexity than Force and Cluster, it will scale worse when the amount of variables increases.

After all, it turned out that it is difficult to comprehensively measure the quality of the
static methods because none of the investigated configuration problems could be successfully compiled without applying dynamic methods. Therefore, measuring or even proving the superiority of one of the methods was not possible. The two introduced experiments (and numerous further experiments) allow to conclude that the MSR ordering performs best in the context of the investigated configuration problems.

7.4 On the Impact of Dynamic Variable Ordering

As mentioned in the previous section, the application of static methods alone does not allow to compile any of the given configuration problems in reasonable time. Therefore, the dynamic methods introduced in Section 6.1.2 will be applied during (and/or after) the construction of the feasible design space. The application of dynamic methods (also: dynamic optimization) is controlled according to Algorithm Z.11. Note that the parametrization can only be specified once before the construction starts. For each method the window size and the number of maximal iterations must be specified. Given that, the methods’ application is controlled by two further parameters: firstly, the number of constraints to compile before re-ordering (also: offset) and, secondly, the size factor—whenever the growth of the MDD’s size exceeds this value, dynamic methods are triggered. In order to illustrate the effect of the different methods, three experiments will be introduced in the following. All computations were carried out using one core of an Intel Xeon E5-2643 processor at 3.3 GHz with 64 GB RAM.

The first experiment includes the construction of P6’s feasible design space, starting with Cluster. The offset is set to zero and the size factor is three. Then, each dynamic method is applied with two different window sizes and one maximal iteration—see Figure 7.7. Here, next to the dynamics of memory consumption (red line), the legends depict the chosen method, the overall construction time of MDD(F) in seconds, the size of MDD(F), and the total number of re-ordering events (triggered by size factor). Based on that, the dynamic methods’ quality can be measured by both the run-time and the memory consumption of MDD(F). Increasing the window size of Permute or Partial dramatically increases the run-time. While Permute5 requires 62 seconds, Permute7 requires 6217.3 seconds, i.e. a factor of ≈ 1000. In contrast to that, Sifting50 is only approx. 5.5 times slower than Sifting 10—reflecting the linearity within the Sifting method’s complexity (determined by the window size). Of course, the run-time of the dynamic methods cannot be derived from the window size only—the run-time is also most heavily determined by the number of nodes and branches within the diagram (cf. LevelSwap in Section 5.1.2.3). From a memory consumption point of view, the best result is achieved by Partial15: 0.56 MB—this result, however, comes with high computational costs in terms of the run-time: almost 13 hours. The construction is much faster when using Sifting10, but the size of MDD(F) is 2.7/0.56 ≈ 4.5 times larger. The application of Permute7 demonstrates that increased window sizes do not necessarily result in a better variable ordering of MDD(F): 1.82 MB > 0.98 MB.
Figure 7.7: Different dynamic variable ordering techniques applied during the construction of P6’s feasible design space (starting with MSR ordering). Vertical gray lines indicate dynamic re-ordering events.

The setting of the second experiment, which was applied to P1, is as follows: The MDD software starts constructing the feasible design space with MSR as initial variable ordering. The offset is set to 1200 and the compilation sequence enhanced by Sortlevel, accordingly in all cases the memory consumption was 14.58 MB. Based on that, each dynamic method is applied once (maximal iterations = 1) using three different window sizes. The results—namely ‘Final size’, ‘Compression ratio’, and ‘Run-time’—are shown in Table 7.3. Here, in contrast to the first experiment, increasing the window of Permute allows to increase
7.4. On the Impact of Dynamic Variable Ordering

Table 7.3: Comparison of different dynamic variable ordering methods applied to P1

<table>
<thead>
<tr>
<th>Method</th>
<th>Initial Size [MB]</th>
<th>Final Size [MB]</th>
<th>Compression ratio</th>
<th>Run-time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sifting10</td>
<td>14.58</td>
<td>2.25</td>
<td>6.48</td>
<td>2</td>
</tr>
<tr>
<td>Sifting30</td>
<td>14.58</td>
<td>1.76</td>
<td>8.27</td>
<td>12</td>
</tr>
<tr>
<td>Sifting50</td>
<td>14.58</td>
<td>1.73</td>
<td>8.40</td>
<td>65</td>
</tr>
<tr>
<td>Permute3</td>
<td>14.58</td>
<td>2.67</td>
<td>5.44</td>
<td>3</td>
</tr>
<tr>
<td>Permute5</td>
<td>14.58</td>
<td>0.99</td>
<td>14.71</td>
<td>13</td>
</tr>
<tr>
<td>Permute7</td>
<td>14.58</td>
<td>0.58</td>
<td>24.96</td>
<td>1475</td>
</tr>
<tr>
<td>Partial5</td>
<td>14.58</td>
<td>1.07</td>
<td>13.61</td>
<td>3</td>
</tr>
<tr>
<td>Partial7</td>
<td>14.58</td>
<td>0.81</td>
<td>17.91</td>
<td>9</td>
</tr>
<tr>
<td>Partial10</td>
<td>14.58</td>
<td>0.63</td>
<td>22.97</td>
<td>177</td>
</tr>
</tbody>
</table>

Table 7.4: Three optimization strategies marked by different window sizes s and different numbers of iterations per dynamic method

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Window size / Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sifting</td>
</tr>
<tr>
<td></td>
<td>s = 1</td>
</tr>
<tr>
<td>O1</td>
<td>3/4</td>
</tr>
<tr>
<td>O2</td>
<td>3/4</td>
</tr>
<tr>
<td>O3</td>
<td>3/2</td>
</tr>
</tbody>
</table>

the compression (given by ‘Initial Size’ / ‘Final Size’) as well—up to a factor of almost 25. After all, Partial7 performs best in this experiment, achieving relatively high compression in very short time (9 seconds).

In order to determine a generic parameter set to dynamically optimize all given configuration problems, the following experiment is carried out. It comprises the construction of the feasible design space of each configuration problem, starting with MSR ordering, and applying three optimization strategies of different strength (according to the parameter sets in Table 7.4) during the compilation process. For example, O2 includes the combined application of Sifting3, Sifting5, Sifting40, Permute4, and Partial5, each of them with maximal four iterations. While compiling each problem, the sequence is enhanced by Bestfit300 and re-orderings are triggered by a size factor of two. Considering O1, additionally after each 10th dynamic optimization run, Sifting30, Permute6, Partial9—each with 4 iterations—are applied. Note that for P4 also the Cluster static ordering is evaluated.

The impact of the different optimization strategies is given in Table 7.5. Here, the run-times and the size of MDD(F) are given for each configuration problem and optimization strategy, respectively. The strongest strategy, i.e. O1, performs best when considering the size of MDD(F)—except for P6. But, the set of strong parameters causes considerably
more computational effort, see for example P5—111:09:22 hours $\approx$ 4.5 days, or P4\texttt{Cluster}—378:46:43 hours $\approx$ 2 weeks. These examples demonstrate that the run-time of the dynamic methods does not only depend on the size of the windows but also on the number of nodes (and the size of the MDD). The run-time of the strategies $O_2$ and $O_3$ is considerably lower, but MDD($F$) consumes more memory. During the application of $O_2$ to P5 and P4\texttt{Cluster}, MDD(int.) exceeds the memory limitation of 1024 MB. As mentioned before, considering configuration problem P4, two different initial variable orderings are evaluated. Applying $O_1$ combined with \texttt{Cluster} yields a better memory consumption (89.26 MB) than starting with MSR (189.5 MB), but requires a higher run-time (378:46:43 hours in contrast to 65:05:42 hours). Finally, applying $O_3$ to P4\texttt{Cluster} allows to construct MDD($F$) consuming 877.5 MB. The high memory consumption causes a run-time of 24:03:28 hours—much higher than $O_3$ applied to all other configuration problems starting with MSR.

Summing up, the parameter set to control dynamic methods is given by: the offset, the window size, the number of maximal iterations, and the size factor. There is no straightforward strategy (for example based on the number of constraints) allowing to derive the optimal parametrization of the dynamic methods—this mostly depends on the structure of the given configuration problem. In general, the goal is to find a trade-off between the memory consumption of MDD($F$) and the run-time of the construction process. The set of conducted experiments in the context of the automotive configuration problems allows to conclude that combining the static MSR approach along with strategy $O_2$ or $O_3$ yields the best results. Of course, when searching for an improved ordering, the strength of the dynamic strategy can be further increased (cf. window size, size factor, or offset).

### 7.5 On the Impact of the Constraint Compilation Sequence

This section introduces the results from applying the methods to enhance the constraint compilation sequence (see Section 6.2). Next to the \textit{Original} sequence from the database,
three approaches were evaluated: NW, Sortlevel, and Bestfit. These methods aim at keeping the memory consumption during the construction small, which, in turn, increases the performance of dynamic variable ordering techniques. The calculation of the NW sequence is carried out by the pre-processing software of the toolchain. The corresponding run-time varies from 1.01 seconds (P6) to 7.03 seconds (P3) and mainly depends on the number of constraints, domains, and clusters of the problem (see Table 7.1 and Figure 6.4).

The following two experiments are introduced to demonstrate the impact of the different sequences.
Table 7.6: Performance of Bestfit during the construction of the feasible design space

<table>
<thead>
<tr>
<th>Product</th>
<th>Run-time [s]</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bestfit150</td>
<td>Bestfit300</td>
<td>Bestfit600</td>
<td>Memorized</td>
</tr>
<tr>
<td>P1</td>
<td>195.39</td>
<td>218.79</td>
<td>308.40</td>
<td>5.53</td>
</tr>
<tr>
<td>P2</td>
<td>241.70</td>
<td>342.26</td>
<td>458.78</td>
<td>4.72</td>
</tr>
<tr>
<td>P3</td>
<td>124.36</td>
<td>244.32</td>
<td>1085.71</td>
<td>3.33</td>
</tr>
<tr>
<td>P4</td>
<td>486.37</td>
<td>710.20</td>
<td>1247.50</td>
<td>8.25</td>
</tr>
<tr>
<td>P5</td>
<td>1134.50</td>
<td>1475.93</td>
<td>2610.19</td>
<td>15.17</td>
</tr>
<tr>
<td>P6</td>
<td>12.63</td>
<td>22.52</td>
<td>31.44</td>
<td>1.46</td>
</tr>
</tbody>
</table>

In the first experiment, the sequences are evaluated during the construction of the feasible design space of all products. For this purpose, optimized variable orderings are used—up to now, theses orderings cause the least memory consumption (see Table 7.8 and Figure 7.10). The dynamics of the memory consumption are shown in Figure 7.8. When considering the size of the intermediate MDD representations only, Bestfit150 outperforms the other sequences in all cases. But the dynamic calculation of the Bestfit sequence increases the run-time of the overall construction process. Only if the size exceeds a certain limit during the construction—like NW at P1, or Original at P2—Bestfit is faster. In all cases, the NW curve ascends more slowly than the Original curve. However, after having compiled a certain number of constraints, the NW sequence causes a considerable growth of the memory consumption. Considering P5, the size of MDD(int.) exceeds the limitation of 1024 MB when applying the Original or NW sequence. The Sortlevel curve is always below the NW and Original curve, thus applying this sequence results in the fastest run-time (except for P6). This experiment demonstrates that a good variable ordering alone is not sufficient to successfully construct the feasible design space representation because the size of MDD(int.) can exceed the memory limitation.

In the second experiment, the run-times of the Bestfit method using three different window sizes (150, 300, and 600) are compared. Again, the variable ordering is not changed during the experiment and the best variable ordering is used. In addition, the memorized sequence from Bestfit300 is applied during the construction. The run-times are given in Table 7.6. Increasing the window size of Bestfit increases the run-time of the construction process. However, when MDD(F) is re-constructed—for example, when the constraints within the BOM or the AP are changed—the memorized sequence can be applied, which results in a significant speed-up.
7.6. On the Verification

The final section of this chapter presents the results and the performance of the verification process. In the following, two different techniques will be applied: the MDD-based approach (cf. Section 5.1.3) and the SAT-based approach (cf. Section 5.2). While the MDD-based technique has been developed within this research project and in accordance with the formal model (cf. Chapter 4), the model for the SAT-based has been developed to validate the results of the MDD-based technique. Note that the SAT-based technique only comprises the verification of (4.33) and (4.36).

Both techniques revealed that the feasible design space $F$ is not empty considering any of the investigated products. In addition, for each product the SAT-based approach yields the same set of slow-moving items in the BOM as the MDD-based technique.

Table 7.7 shows the run-times of the techniques—in addition, information on the cardinality of the constraints within the BOM is given. All computations were carried out...
Table 7.9: Performance of the MDD-based technique for the verification of further consistency requirements

<table>
<thead>
<tr>
<th>Product</th>
<th>Attributes in the CD</th>
<th>Run-time [s]</th>
<th>Attributes</th>
<th>Part types</th>
<th>Assembly steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>(4.34)</td>
<td>(4.37)</td>
<td>(4.38) (4.39) (4.40)</td>
</tr>
<tr>
<td>P1</td>
<td>989</td>
<td>9.032</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P2</td>
<td>1001</td>
<td>4.477</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P3</td>
<td>943</td>
<td>0.328</td>
<td>0.031</td>
<td>0.045</td>
<td>0.172</td>
</tr>
<tr>
<td>P4</td>
<td>1240</td>
<td>2.730</td>
<td>0.016</td>
<td>0.202</td>
<td>1.154</td>
</tr>
<tr>
<td>P5</td>
<td>1116</td>
<td>19.687</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>P6</td>
<td>781</td>
<td>0.140</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

using one core of the 3.3 GHz Intel Xeon E5-2643 processor with 64 GB of RAM. Note that for the MDD-based approach, the best variable ordering with the least memory consumption was used (see Table 7.8 and Figure 7.10). The number of constraints within the BOM alone does not allow to draw conclusions on the run-time of the verification process (see, for example, P1 and P3). Still, compared to the number of composite constraints within the BOM and the AP—only available for P3 and P4—this number is relatively high, therefore, the run-time of the verification of this consistency rule has the highest expressiveness.

Both the MDD-based and the incremental SAT-approach outperform the sequential technique. Except for P1, the MDD-based technique performs better than the incremental SAT-technique. Whenever the MDD representation of the feasible design space exceeds a certain limit, for example P1, P4, or P5, the run-time of the SAT-Solver is only slightly higher or—in case of P1—even better\(^1\). When taking into account both the complexity of the configuration problems and the number of constraints in the BOM (i.e. the number of SetIntersect operations in the MDD case), both techniques reveal the slow-moving items in a very short time.

Table 7.8 shows the cardinality of the design space \(S\) and of the feasible design space \(F\). This information can be extracted directly from the corresponding MDD representations. Even though the constraints \(c_i \in C_{CD}\) significantly reduce the number of configurations, the remaining number of valid configurations is still far too high for explicit storage. \(MDD(F)\) of the problem P5 consumes most memory (\(\approx 120 \text{ MB}\)), followed by P4, P1, P2, P3, and P6. The memory consumption (and the number of nodes) does not only affect the run-time of the verification process but also the dynamic optimization during the construction of the feasible design space (cf. Section 7.4).

\(^1\) According to the information on the website of the SAT-Solver [1], a C++ implementation is about three times faster than the Java-based implementation. Bearing this in mind, the run-time of the SAT-based technique would also be better considering P4 and P5.
Table 7.9 shows the run-time of the MDD-based approach applied to further consistency requirements. When considering the verification of (4.34), the size of the corresponding MDD representations—again—strongly affects the run-time. And, in contrast to the number of constraints within the BOM, the cardinality of the sets of attributes $A$ is quite similar considering the range of products. The mean number of ordinary constraints that are linked to a part type (cf. composite constraint, Definition 4.9) is given by 1.07 (P3) and 1.06 (P4)—hence, in most cases only one constraint is linked to a part type. In contrast to that, the mean number of ordinary constraints linked to an assembly step is given by 1.69 (P3) and 2.04 (P4). However, the corresponding sets can be represented very efficiently by using MDDs and SetUnion operations, which leads to the low run-times.
8 Conclusions

8.1 Summary

In the first chapter, the setting of this research project was introduced: a new IT system replacing paper to control the assembly process of cars. While this new paradigm comes with numerous advantages, like the reduction of complexity for the operators of the assembly line, it also bears the risk of systematic deviations of the assembly process induced by errors within the input data—whose occurrence is favored by various circumstances. In general, automotive manufacturing has become a complex task due to several reasons: increasing competition, the pressure to be innovative coupled with the rapid technological progress, the application of simultaneous engineering (SE), or the rising demand for customizable products, just to name a few. This complexity, accordingly, is also reflected by the information flows and by the data stored in the corresponding IT systems. But, the higher the complexity within the data, the higher the risk for inconsistencies. Therefore, in the course of this thesis, a new method to ensure consistency within the involved data, i.e. the configuration data (CD), the bill of materials (BOM), and the assembly plan (AP) has been developed.

In order to illustrate the manifoldness and complexity of a car manufacturer’s activities, the second chapter took a closer look at the automotive manufacturing domain. The input data of the newly introduced system originates from the development cycle of cars, therefore, next to a brief overview of the car’s evolution through history, a summary of the product development process (PDP) was given. During the development of a car, the application of SE ought to reduce the time to market (TTM). As a side effect of SE, the amount of communication will increase—bearing the risk of misinterpretation and, in the worst case, the occurrence of data errors—these effects were explained in Section 2.2. Finally, since the new system is designed to get installed within the assembly environment, Section 2.3 introduced the production process focusing on the assembly of cars.

The focus of the third chapter was on data quality (DQ). In Section 3.1, a classification of data quality problems and an introduction to data errors was provided. These terms were illustrated by way of several examples. Furthermore, in Section 3.2, the concept of data quality dimensions (DQDs) was explained—including the introduction of the most important state-of-the-art dimensions. Such dimensions, like completeness, are the basis for the development of metrics which, among others, allow to measure the quality of data. Focusing on the total data quality management (TDQM) approach, the final Section 3.3 briefly touched the data quality management topic.

The fourth chapter introduced the formal model of input data of the eWBK system. This chapter also described the mapping, i.e. the way of how the information on the screens
of the eWBK system is acquired. This is why in the course of the project the most important question was how to ensure consistency among the constraints of the involved data. Based on the mapping and the formal model of the data, the consistency requirements of the eWBK system were developed in Section 4.6.

Chapter 5 introduced a new technique to verify the consistency requirements. For this, multi-valued decision diagrams (MDDs) are particularly suited considering the multi-valued domains of the configuration problems and of the constraints within the BOM and the AP. Coupled with the formal model of the input data and the consistency requirements, the MDD-based technique allows for a straightforward verification by applying simple set operations, like \textit{SetMinus} or \textit{SetIntersect}. The most important programming paradigms of the MDD software were presented in Section 5.1.2. Finally, in order to validate the MDD-based approach, a model for the SAT-based technique was introduced in Section 5.2. Next to validation, this also allowed to compare the performance of both techniques.

Considering some of the investigated configuration problems, it turned out that the MDD representations of the feasible design space consumed too much memory. In order to overcome this effect, methods to improve the variable ordering, both static and dynamic, were introduced in Section 6.1. Next to already established approaches from literature, mass-spring-relaxation (MSR) was introduced as a new static method. In addition, methods to enhance the sequence of compiling the constraints were presented in Section 6.2—including two new methods, i.e. \textit{Sortlevel} and \textit{Bestfit}.

Finally, Chapter 7 introduced the results of applying the presented verification techniques to the real-world automotive product data. Its first two sections inclosed both the description of the toolchain and the introduction of the automotive configuration problems. Then, in Section 7.3, the impact of static variable ordering methods was illustrated—it turned out that in the context of the investigated configuration problems, the MSR approach performs best. Section 7.4 focused on the impact of the dynamic variable ordering methods. Even with a good initial variable ordering, the feasible design space could not be constructed in some cases. Only a combination of all dynamic methods allowed to successfully compile the constraints of all problems. In addition, Section 7.5 demonstrated that the new methods to enhance the compilations sequence keep the memory consumption of intermediate MDD representations applicable—this is important since intermediate MDD representations can exceed memory limitations. Finally, in Section 7.6, the validation and the performance of the verification techniques were presented.

8.2 Outlook and Future Work

Combining the newly presented performance optimization methods along with established methods from literature allowed to apply the MDD-based verification technique for highly customizable products of the AUDI AG. However, the MDD-based representation of the feasible design space could also be used in other applications: for example backtrack-free
8.2. Outlook and Future Work

product configuration (cf. [27]). Once having managed to compile the constraints of the configuration problem, various applications and analysis issues could be realized by implementing the corresponding operations within the MDD software.

In order to successfully construct the feasible design space of larger problems, the application of dynamic methods could be further improved by implementing a flexible control of their application: dynamically adapting the window size, defining abortion criteria, or changing the sequence of their application, among others.

Furthermore, it would be interesting to investigate the effect of reducing the complexity of large configuration problems by splitting the domain of a certain variable. This would allow, for example, to investigate the inclosed variants or the destination countries separately. As a consequence, the number of configuration problems would increase, which, in turn, could be counteracted by increasing the number of computing devices.

Parallelization could not only be achieved on the model level but also within the MDD software—for example, to decrease the computational effort of the Bestfit method (cf. [61]).

In general, various methods—not specifically focusing on consistency among constraints—could be implemented to increase the overall DQ of automotive product data: for example, the meta-rules approach for the configuration data as proposed in [11].
List of Figures

1.1 Replacing paper by an IT system ............................................ 2
1.2 Information flow of the eWBK system ..................................... 3
2.1 Product range and automotive product data .............................. 8
2.2 Internal and external circumstances adding complexity to the business of automotive manufacturers ........................................... 9
2.3 Overview of the product development process (PDP) .................. 10
2.4 Sequential and parallel execution of activities during the PDP ....... 12
2.5 Interdependence of processes .................................................. 13
2.6 PPC and the main facilities of automotive manufacturing .......... 16
2.7 Scheme of an assembly line ................................................... 17
3.1 Classification of data quality problems according to Rahm et al. [55] 23
3.2 TDQM cycle according to Wang [71] ....................................... 29
3.3 Framework conditions of the information flow of the eWBK system . 31
4.1 Different representations of (4.1) ............................................ 35
4.2 Overview of both the relations among the entities of the input data and the responsible business units ................................. 36
4.3 Attributes and domains .......................................................... 38
4.4 Standardized structure of products .......................................... 42
4.5 Organization of assembly steps (red) following the structure of products (left) or the structure of the assembly line (right) ............. 45
4.6 Mapping to generate information on the eWBK-screens ............... 47
4.7 Illustration of the set-based consistency requirement (4.36) ......... 49
5.1 MDD representations of Example 5.1 ....................................... 53
5.2 MDDs encoding sets of configurations ..................................... 54
5.3 Illustration of nodes and branches—cf. the first two levels of Figure 5.2 (a) ................................................................. 55
5.4 Caches .............................................................................. 56
5.5 The LevelSwap of two adjacent levels within an MDD ............... 58
5.6 The SetMinus operation ......................................................... 59
5.7 MDD representation of the feasible design space—MDD(F) ........ 60
6.1 Memory consumption during the construction of the feasible design space with and without applying performance optimization measures .... 63
6.2 Two different variable orderings of ROBDDs representing (4.1) ... 64
6.3 Constraint graph of a medium-sized configuration problem with $|V| = |D| = 100$ and $|C| = 400$. The red node reflects the central variable. .... 66
6.4 Clustering within the constraint graphs of the automotive configuration problems. While the x-axis contains the IDs of the clusters, the y-axis depicts the number of contained variables. The legend specifies the percentage of variables contained in the largest cluster. ................................................................. 67
6.5 Constraint graph of a real-world automotive configuration problem (P5) with $|V| = |D| = 160$ and $|C| = 3844$ ............................... 68
6.6 Application of Force w.r.t. four different, randomly chosen initial positions of the variables ......................................................... 69
6.7 Illustration of the MSR approach ........................................... 71
7.1 Toolchain for the verification approach ................................. 80
7.2 Dynamics of memory consumption and the number of nodes ........................................ 81
7.3 Overview of the size of the constraints’ scopes. The legend depicts the percentage of binary constraints. ................................................................. 82
7.4 Illustration of three different static variable ordering methods applied to the configuration problem P5. In addition, the variable ordering obtained from applying dynamic methods is shown. ......................................................... 84
7.5 Comparison of the four different static variable orderings for the configuration problem P6. After having compiled 1000 constraints, the dynamic ordering technique *Permute5* is applied. Vertical gray lines indicate dynamic re-ordering events. ................................................................. 85
7.6 Comparison of the four different static variable orderings applied to the configuration problem P2. For this experiment dynamic reordering is disabled, therefore, the set of constraints to compile is restricted to 300. ........................................ 86
7.7 Different dynamic variable ordering techniques applied during the construction of P6’s feasible design space (starting with *MSR* ordering). Vertical gray lines indicate dynamic re-ordering events. ................................................................. 88
7.8 Application of different constraint compilation sequences. Dynamic variable ordering methods were disabled in this experiment. ........................................ 91
7.9 Illustration of the *MSR* static variable ordering for each configuration problem ........ 95
7.10 Illustration of the best variable ordering, obtained by dynamic optimization, for each configuration problem ................................................................. 95
List of Tables

3.1 Data errors demonstrated by the example of the federal states of Germany . . . 24
3.2 DQ categories and dimensions according to Wang et al. [72] . . . . . . . . . 28
4.1 Truth table of (4.1) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35
4.2 Properties defining a car . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
4.3 List of parts within the bill of materials . . . . . . . . . . . . . . . . . . . . . . 41
4.4 Aggregation of parts within part types . . . . . . . . . . . . . . . . . . . . . . . 43
4.5 Visualization properties within assembly steps to acquire the content of the
eWBK-screens . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 46
6.1 Constraints for the MSR example . . . . . . . . . . . . . . . . . . . . . . . . . . 72
7.1 Dimensions of the automotive configuration problems . . . . . . . . . . . . . . 81
7.2 Performance of the static variable ordering methods . . . . . . . . . . . . . . . 83
7.3 Comparison of different dynamic variable ordering methods applied to P1 . . 89
7.4 Three optimization strategies marked by different window sizes $s$ and different
numbers of iterations per dynamic method . . . . . . . . . . . . . . . . . . . . . . . 89
7.5 Impact of the optimization strategies . . . . . . . . . . . . . . . . . . . . . . . . 90
7.6 Performance of Bestfit during the construction of the feasible design space . . 92
7.7 Performance of the techniques for the detection of slow-moving items (4.36) . . 93
7.8 Cardinality of both the design space $S$ and the feasible design space $F$. In
addition, the memory consumption of MDD($F$) and the corresponding number
of nodes are given. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93
7.9 Performance of the MDD-based technique for the verification of further consistency
requirements . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 94
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD</td>
<td>Binary decision diagram</td>
</tr>
<tr>
<td>BOM</td>
<td>Bill of materials</td>
</tr>
<tr>
<td>CD</td>
<td>Configuration data</td>
</tr>
<tr>
<td>CSP</td>
<td>Constraint satisfaction problem</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed acyclic graph</td>
</tr>
<tr>
<td>DCM</td>
<td>Data control model</td>
</tr>
<tr>
<td>DQ</td>
<td>Data quality</td>
</tr>
<tr>
<td>DQD</td>
<td>Data quality dimension</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic control unit</td>
</tr>
<tr>
<td>eWBK</td>
<td>Elektronische Wagenbegleitkarte</td>
</tr>
<tr>
<td>ER</td>
<td>Entity-relationship</td>
</tr>
<tr>
<td>FMEA</td>
<td>Failure mode and effects analysis</td>
</tr>
<tr>
<td>FTA</td>
<td>Fault tree analysis</td>
</tr>
<tr>
<td>GB</td>
<td>Gigabyte</td>
</tr>
<tr>
<td>GHz</td>
<td>Gigahertz</td>
</tr>
<tr>
<td>GIGO</td>
<td>Garbage in, garbage out</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-machine interface</td>
</tr>
<tr>
<td>INI.FAU</td>
<td>Ingolstadt Institute der Friedrich-Alexander Universität Erlangen-Nürnberg</td>
</tr>
<tr>
<td>IP</td>
<td>Information product</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>LCM</td>
<td>Least common multiple</td>
</tr>
<tr>
<td>LES</td>
<td>Linear equation system</td>
</tr>
<tr>
<td>M2M</td>
<td>Machine-to-machine interface</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>MCL</td>
<td>Markov cluster algorithm</td>
</tr>
<tr>
<td>MDD</td>
<td>Multi-valued decision diagram</td>
</tr>
<tr>
<td>MSR</td>
<td>Mass-spring-relaxation</td>
</tr>
<tr>
<td>PDP</td>
<td>Product development process</td>
</tr>
<tr>
<td>PPC</td>
<td>Production planning and control</td>
</tr>
<tr>
<td>R&amp;D</td>
<td>Research and development</td>
</tr>
<tr>
<td>RAM</td>
<td>Random-access memory</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>ROBDD</td>
<td>Reduced ordered binary decision diagram</td>
</tr>
<tr>
<td>ROMDD</td>
<td>Reduced ordered multi-valued decision diagram</td>
</tr>
<tr>
<td>SANET</td>
<td>Sensor and actor network</td>
</tr>
<tr>
<td>SAT</td>
<td>Boolean satisfiability problem</td>
</tr>
<tr>
<td>SE</td>
<td>Simultaneous engineering</td>
</tr>
<tr>
<td>TDQM</td>
<td>Total data quality management</td>
</tr>
<tr>
<td>TPS</td>
<td>Toyota production system</td>
</tr>
<tr>
<td>TQM</td>
<td>Total quality management</td>
</tr>
<tr>
<td>TTM</td>
<td>Time to market</td>
</tr>
<tr>
<td>UID</td>
<td>Unique identifier</td>
</tr>
</tbody>
</table>

**Milestones of the product development process:**
(Order of occurrence, see Figure 2.3)

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS</td>
<td>Project start</td>
</tr>
<tr>
<td>PF</td>
<td>Project feasibility</td>
</tr>
<tr>
<td>CoD</td>
<td>Concept decision</td>
</tr>
<tr>
<td>DD</td>
<td>Design decision</td>
</tr>
<tr>
<td>La</td>
<td>Launch</td>
</tr>
<tr>
<td>IB</td>
<td>Initial batch</td>
</tr>
<tr>
<td>SOP</td>
<td>Start of production</td>
</tr>
<tr>
<td>ITM</td>
<td>Introduction to the market</td>
</tr>
<tr>
<td>EOP</td>
<td>End of production</td>
</tr>
</tbody>
</table>
Bibliography


Recursion is a central idea of computer science since it is strongly related to the divide-and-conquer paradigm: A considerably large problem is divided into smaller problems of the same structure. Once having described the solution process for this kind of problem, it can be applied to all recurring sub-problems due to their similar structure.

Recursive functions are functions which are called within their own definition—this fundamental principle can be applied, for example, to calculate the factorial:

\begin{align*}
\text{fac}(n) := \begin{cases} 
1 & \text{for } n = 0 \\
 n \cdot \text{fac}(n - 1) & \text{for } n > 0
\end{cases} \quad (Z.1)
\end{align*}

For example, \( \text{fac}(5) \) will produce

\[5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120.\]

Here, \( n = 0 \) breaks the chain of recursion, thus it is called terminal case (also: base case).

This principle is applied in the MDD software to traverse all paths within the diagram—required while carrying out operations like \text{SetMinus}, \text{SetIntersect}, and \text{SetUnion}. To illustrate this, the \text{SetMinus} operation is described in the following.
Algorithm Z.1 The SetMinus operation (Section 5.1.2.2)

Require: Two MDDs represented by their root nodes \( n_1 \) and \( n_2 \)
Require: Current level of operation: \( L \)
Require: Vector of vectors storing intermediate result w.r.t. \( L \): \( \text{vvec} \)

1: if \((n_1 == 0) \lor (n_2 == n_1)\) then
2: \( \text{return } 0 \)
3: end if
4: if \( n_2 == 0 \) then
5: \( \text{return } n_1 \)
6: end if
7: if \((\text{cacheElem } \leftarrow \text{operations-cache}.\text{find}(n_1, n_2)) \neq 0\) then
8: \( \text{return } \text{cacheElem}.$\text{result}() \)
9: end if
10: vec \(-\text{vvec}[L] \)
11: \( b_1 \leftarrow n_1.\text{branch}() \)
12: \( b_2 \leftarrow n_2.\text{branch}() \)
13: while \((b_1 \land b_2)\) do \{Iterate over branches, compute intermediate results, and recursively call SetMinus\}
14: if \( b_1.\text{val}() < b_2.\text{val}()\) then
15: \( \text{vec}.\text{storeBranchelem}(b_1.\text{val}(), b_1.\text{child}()) \)
16: \( b_1 \leftarrow b_1.\text{next}() \)
17: else if \( b_1.\text{val}() > b_2.\text{val}()\) then
18: \( b_2 \leftarrow b_2.\text{next}() \)
19: else
20: if \( \text{TerminalCase}(L) == \text{false} \) then \{Terminal case is identified via \( L \}\}
21: \( \text{tmpNode } \leftarrow \text{SetMinus}(b_1.\text{child}(), b_2.\text{child}(), L+1, \text{vvec}) \)
22: if \( \text{tmpNode} \neq 0 \) then
23: \( \text{vec}.\text{storeBranchelem}(b_1.\text{val}(), \text{tmpNode}) \)
24: end if
25: else if \( \text{TerminalCase}(L) == \text{true} \) then
26: ;;
27: end if
28: \( b_1 \leftarrow b_1.\text{next}() \)
29: \( b_2 \leftarrow b_2.\text{next}() \)
30: end if
31: end while
32: for \((; b_1 \neq 0; b_1 \leftarrow b_1.\text{next}())\) do
33: \( \text{vec}.\text{storeBranchelem}(b_1.\text{val}(), b_1.\text{child}()) \)
34: end for
35: Construct node Ret from all branches \( \in \text{vec} \)
36: \( \text{return } \text{Ret} \)
Algorithm Z.2 The LevelSwap operation (Section 5.1.2.3)

**Require:** Levels to swap: \( i \), (and \( i + 1 \), respectively)

1: Create empty list \( L \)
2: \( \textbf{for all} \) nodes \( n \in \text{level } i \) \( \textbf{do} \)
3: \( \textbf{for all} \) branches \( b_i \in n.\text{branches()} \) \( \textbf{do} \)
4: \( \textbf{for all} \) branches \( b_{i+1} \in b_i.\text{child().branches()} \) \( \textbf{do} \)
5: Construct list element \( l \) \{Switch values of branches while storing elements\}
6: \( l_1 \leftarrow b_{i+1}.\text{val()} \)
7: \( l_2 \leftarrow b_i.\text{val()} \)
8: \( l_3 \leftarrow b_{i+1}.\text{child()} \)
9: \( L.\text{append}(l) \)
10: \( \textbf{end for} \)
11: \( \textbf{end for} \)
12: Sort list elements \( \in L \) w.r.t. \( l_1 \) and \( l_2 \)
13: \( \textbf{for all} \) elements \( l \in L \) \( \textbf{do} \)
14: \( \textbf{if} \) \( l_1 \) is new value \( \textbf{then} \)
15: Create node tmpNode
16: \( \textbf{end if} \)
17: Add new branch \( b \) with \( b.\text{val} = l_2 \) to tmpNode
18: \( \textbf{end for} \)
19: \( \textbf{if} \) \( \text{node-cache}.\text{find(tmpNode)} \neq 0 \) \( \textbf{then} \)
20: \( \text{;;} \)
21: \( \textbf{else} \)
22: \( \text{node-cache}.\text{add(tmpNode)} \)
23: \( \textbf{end if} \)
24: \( \textbf{end for} \)
Algorithm Z.3 Construction of the feasible design space (Section 5.1.3)

Require: Configuration problem \( \langle V, D, C_{CD} \rangle \)

Require: Bundle parameter \( b \geq 1 \)

1: Construct MDD(S)
2: MDD(int.) \( \leftarrow \) MDD(S)
3: while \( C_{CD} \neq \emptyset \) do
4: \( MDD_0 \leftarrow \emptyset \)
5: for \( i = 1 ; i \leq b ; i++ \) do
6: \{ Construct MDD representation of a bundle of constraints \}
7: Choose constraint \( c \in C_{CD} \)
8: Construct MDD\( (c) \)
9: \( MDD_i \leftarrow \text{SetUnion}(MDD_{i-1}, \text{MDD}(c)) \)
10: \( C_{CD} \leftarrow C_{CD} \setminus c \)
11: end for
12: \( \text{MDD(int.)} \leftarrow \text{SetMinus}(\text{MDD(int.)}, MDD_i) \)
13: end while
14: \( \text{MDD}(F) \leftarrow \text{MDD(int.)} \)
Algorithm Z.4 Sequential approach (Section 5.2)

Require: Configuration problem \((V, D, C, CD)\)

Require: Set of parts \(P\)

1: Introduce binary variables \(a_i\) according to (5.6)
2: Build set of binary, linear constraints \(C_1\) according to (5.7)
3: Build set of binary, linear constraints \(C_2\) according to (5.8)
4: \(C \leftarrow C_1 \cup C_2\)
5: for all \(c_p \in P\) do
6: Build binary, linear constraint \(c_{\text{tmp}}\) of \(c_p\) according to (5.9)
7: \(C_{\text{SAT}} \leftarrow C \cup c_{\text{tmp}}\)
8: if SAT\((C_{\text{SAT}}) == \text{infeasible}\) then
9: \(p\) is a slow-moving item
10: end if
11: end for

Algorithm Z.5 Incremental approach (Section 5.2)

Require: Configuration problem \((V, D, C, CD)\)

Require: Set of parts \(P\) within the bill of materials (BOM)

1: Introduce binary variables \(a_i\) according to (5.6)
2: Build set of binary, linear constraints \(C_1\) according to (5.7)
3: Build set of binary, linear constraints \(C_2\) according to (5.8)
4: Introduce binary variables \(c_p\) according to (5.10)
5: Build set of binary, linear constraints \(C_3\) according to (5.11)
6: \(C_{\text{SAT}} \leftarrow C_1 \cup C_2 \cup C_3\)
7: \(C_{\text{BOM}} \leftarrow \{c_p \mid p \in P\}\)
8: repeat
9: Add constraint of the form (5.12) to \(C_{\text{SAT}}\)
10: Solution \(s \leftarrow \text{SAT}(C_{\text{SAT}})\)
11: for all \(c_p = 1 \in s\) do
12: \(C_{\text{BOM}} \leftarrow C_{\text{BOM}} \setminus c_p\)
13: end for
14: until \((C_{\text{BOM}} == \emptyset) \lor (\text{SAT}(C_{\text{SAT}}) == \text{infeasible}))\)
Algorithm Z.6 Mass-spring-relaxation (MSR) (Section 6.1.1.3)

Require: Configuration problem $\langle V, D, C_{CD} \rangle$

1: for all $v_i, v_j \in V$ do
2: Derive spring constant $K_{i,j}$ according to (6.7)
3: end for

4: for all $v_i \in V$ do
5: Map $v_i$ to arbitrary $x_i \in [0, 1]$
6: end for

7: Derive LES $K \cdot X = F$
8: $\min \leftarrow \infty$
9: for all $i = 0$ to $m - 1$ do
10: for all $j = i + 1$ to $m$ do
11: Set $X' = X$ with $x_i = 0$ and $x_j = 1$
12: Set $F' = F$ with $F_k = 0, k \neq i, j$
13: Solve $K \cdot X' = F'$
14: if $w(X') < \min$ then
15: $\min \leftarrow w(X')$
16: $X_{opt} \leftarrow X'$
17: end if
18: end for
19: end for
20: Transfer natural ordering $\leq$ of $X_{opt}$ to $V$
Algorithm Z.7 The Sifting method (Section 6.1.2)

Require: Configuration problem \( \langle V, D, C_D \rangle \) with \( m := |V| = |D| \)

Require: Either \( \text{MDD}(F) \) or \( \text{MDD}(\text{int.}) \) (cf. Section 5.1.3)

Require: Window size \( s \)

Ensure: \( s \leq m \)

\( \{ \text{Move window from the top to the bottom} \} \)

1: for \( i=0 ; i<(m-1) ; i++ \) do
2: Store memory consumption
3: for \( j=i ; j<(i+s-1) \land j<(m-1) ; j++ \) do
4: \( \text{LevelSwap}(j, j+1) \)
5: Store memory consumption
6: end for
7: Keep arrangement with least memory consumption
8: end for

\( \{ \text{Move window from the bottom to the top} \} \)

9: for \( i=m-1 ; i>0 ; i-- \) do
10: Store memory consumption
11: for \( j=i ; j>(i-s+1) \land j>0 ; j-- \) do
12: \( \text{LevelSwap}(j, j-1) \)
13: Store memory consumption
14: end for
15: Keep arrangement with least memory consumption
16: end for
Algorithm Z.8 The Permute (and Partial) method (Section 6.1.2)

Require: Configuration problem \( \langle V, D, C_D \rangle \) with \( m := |V| = |D| \)
Require: Either MDD\( (F) \) or MDD\( (\text{int.}) \) (cf. Section 5.1.3)
Require: Window size \( s \)
Ensure: \( s \leq m \)

\[
\begin{align*}
\{ \text{Move window from the top to the bottom} \} \\
1: & \text{ for } (i=0 \ ; \ i \leq (m-s) \ ; \ i++) \ do \\
2: & \text{ Store memory consumption} \\
3: & \text{ if } \text{Permute} \text{ then} \\
4: & \text{ Recursively traverse all permutations in window } [i, i+s-1] \text{ using LevelSwap} \\
5: & \text{ end if} \\
6: & \text{ if } \text{Partial} \text{ then} \\
7: & \text{ Recursively traverse subset of permutations in window } [i, i+s-1] \text{ using LevelSwap} \\
8: & \text{ end if} \\
9: & \text{ Keep variable ordering with least memory consumption} \\
10: & \text{ end for} \\
\{ \text{Move window from the bottom to the top} \} \\
11: & \text{ for } (i=(m-s) \ ; \ i \geq 0 \ ; \ i--) \ do \\
12: & \text{ Store memory consumption} \\
13: & \text{ if } \text{Permute} \text{ then} \\
14: & \text{ Recursively traverse all permutations in window } [i, i+s-1] \text{ using LevelSwap} \\
15: & \text{ end if} \\
16: & \text{ if } \text{Partial} \text{ then} \\
17: & \text{ Recursively traverse subset of permutations in window } [i, i+s-1] \text{ using LevelSwap} \\
18: & \text{ end if} \\
19: & \text{ Keep variable ordering with least memory consumption} \\
20: & \text{ end for}
\end{align*}
\]
Appendix 121

Algorithm Z.9 The Sortlevel method (Section 6.2.2)
\[\begin{align*}
\textbf{Require}: & \text{ Variable ordering } O \\
\textbf{Require}: & \text{ Set of constraints } C \text{ with } k := |C|
\end{align*}\]
1: for (i=0 ; i<k ; i++) do
2: Determine levels of \( c_i \) according to (6.24)
3: Determine distance \( d_i \) of \( c_i \) according to (6.25)
4: \( c_i, \text{distance} \leftarrow d_i \)
5: end for
6: Sort constraints \( c_i \in C \) w.r.t. their distance

Algorithm Z.10 The Bestfit method (Section 6.2.3)
\[\begin{align*}
\textbf{Require}: & \text{ Configuration problem } \langle V,D,C_{CD} \rangle \\
\textbf{Require}: & \hat{C}_{CD} \subseteq C_{CD} \text{ and MDD(int.) according to (6.27)} \\
\textbf{Require}: & \text{ Window size } s
\end{align*}\]
1: Counter \( t \leftarrow 0 \)
2: while \( \hat{C}_{CD} \neq C_{CD} \) do
3: \( R \leftarrow C_{CD} \setminus \hat{C}_{CD} \)
4: Sort \( R \) by applying the Sortlevel method according to Algorithm Z.9
5: \{ This has to be carried out only once, or whenever the variable ordering changes \}
6: if \( t \equiv 0 \) then
7: Select first \( s \) constraints of \( R \) to be elements of \( \tilde{R}_t \)
8: else
9: Select first \( s/2 \) constraints of \( R \) and \( s/2 \) uncompiled constraints of \( \tilde{R}_{t-1} \) to be elements of \( \tilde{R}_t \)
10: end if
11: for all \( c \in \tilde{R}_t \) do
12: \( \text{MDD}_{\text{tmp}} \leftarrow \text{SetMinus(MDD(int.), MDD(c))} \)
13: Store memory consumption of \( \text{MDD}_{\text{tmp}} \)
14: Discard \( \text{MDD}_{\text{tmp}} \)
15: end for
16: Identify \( c_{bf} \) with least memory consumption
17: \( \hat{C}_{CD} \leftarrow \hat{C}_{CD} \cup \{c_{bf}\} \)
18: \( \text{MDD(int.)} \leftarrow \text{SetMinus(MDD(int.), MDD(c_{bf}))} \)
19: \( t \leftarrow t + + \)
20: end while
Algorithm Z.11 Control of the dynamic variable ordering methods

Require: Configuration problem $\langle V, D, C_{CD} \rangle$
Require: Offset $n$, i.e. the number of constraints to compile without dynamic optimization
Require: Dynamic variable ordering methods to apply (cf. Section 6.1.2)
Require: Window size for each method
Require: Number of maximal iterations for each method, $it_{max}$
Require: Size factor, $f > 1$

1: $\text{mem}_{old} \leftarrow$ memory consumption of MDD($S$)
2: while true do
3: Construct the feasible design space according to Algorithm Z.3
4: Monitor current memory consumption, i.e. $\text{mem}_{cur}$
5: if ((Compiled constraints $\geq n$) \&\& ($\text{mem}_{cur} \geq f \cdot \text{mem}_{old}$)) then \{Trigger application of dynamic methods\}
6: for all method $\in$ methods \{Sequence: Sifting, Permute, Partial\}
7: for (i=0 ; i $\leq$ $it_{max}$ ; i++) do
8: Apply method
9: Memory consumption = $\text{mem}_{new}$
10: if $\text{mem}_{new} > 0.95 \cdot \text{mem}_{cur}$ then \{Stop application if no significant compression\}
11: Break
12: end if
13: end for
14: end for
15: end if
16: end while