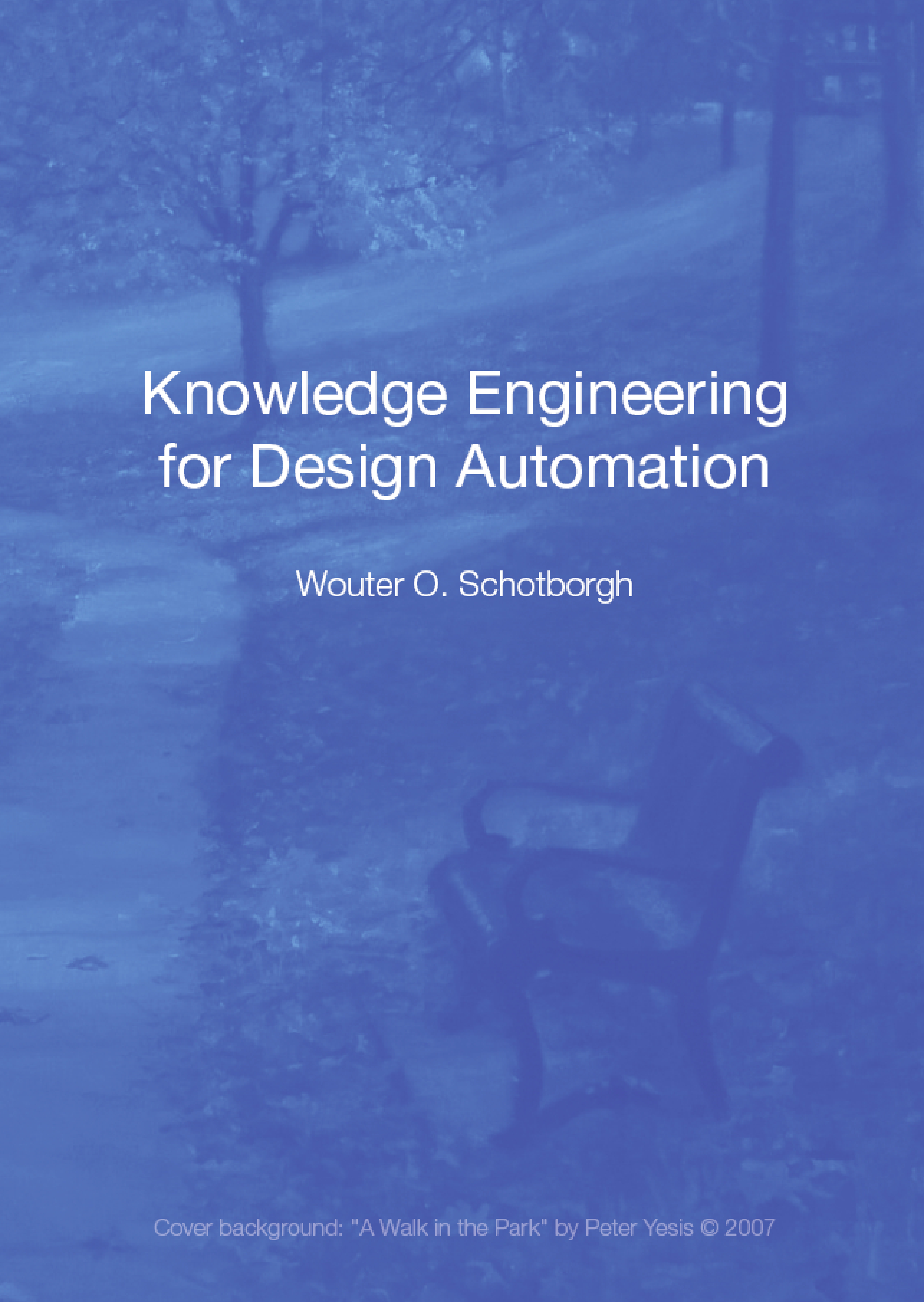


Knowledge Engineering for Design Automation

Wouter O. Schotborgh

Cover background: "A Walk in the Park" by Peter Yesis © 2007

KNOWLEDGE ENGINEERING FOR DESIGN AUTOMATION

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 24 april 2009 om 15.00 uur

door

Wouter Olivier Schotborgh
geboren op 8 juni 1979
te Groningen

Dit proefschrift is goedgekeurd door de promotor
prof. dr. ir. F.J.A.M. van Houten.

ISBN 978-90-365-2801-6

KNOWLEDGE ENGINEERING FOR DESIGN AUTOMATION

PhD Thesis

By Wouter Olivier Schotborgh at the Faculty of Engineering Technology (CTW) of the University of Twente, Enschede, the Netherlands.

Enschede, 24 april 2009

De promotiecommissie:

prof. dr. F. Eising	Universiteit Twente, voorzitter, secretaris
prof. dr. ir. F.J.A.M. van Houten	Universiteit Twente, promotor
prof. dr. ir. A. de Boer	Universiteit Twente
prof. dr. ir. O.A.M. Fisscher	Universiteit Twente
prof. dr. J. van Hillegersberg	Universiteit Twente
prof. D. Brissaud	Grenoble Institute of Technology
prof. dr. K. Shea	Technical University München
prof. dr. T. Tomiyama	Technische Universiteit Delft
dr. G. Still	Universiteit Twente

Keywords: Knowledge Engineering, Engineering Design,
Expert Knowledge, Design Automation

Aan mijn ouders,
aan Kelly

Summary

Engineering design teams face many challenges, one of which is the time pressure on the product creation process. A wide range of Information and Communications Technology solutions is available to relieve the time pressure and increase overall efficiency. A promising type of software is that which automates a design process and generates design candidates, based on specifications of required behavior. Visual presentation of multiple solutions in a “solution space” provides insight in the trends, limitations and possibilities. This higher-level knowledge enables the use of “design intent” and tacit experience knowledge to select the best design for a specific application.

This thesis focuses on software support for (engineering) design processes that use existing technologies and knowledge, with parametric information and quantitative data. This covers continuous and discontinuous parameters, as well as a mix of linear and non-linear equations, logic and fuzzy estimations, for static and dynamic topologies. The scope includes the design of machine elements, product components and product systems.

Academic research has explored the automation of design processes for a wide range of engineering problems, including the scope of this thesis. A variety of theories, frameworks and techniques are developed to automate models of design problems. Sophisticated software support is made possible with advanced functionalities to navigate and explore design solutions. Although the technical feasibility appears to be proved, the intended software support is not present in industry to the extent that it could.

The goal of this thesis is to increase the use of design automation software in industrial environments. The focus lies on efficient development of the models that are required for automation, with the emphasis on expert knowledge for the design creation phase. A method is proposed to acquire the necessary models and determine the software functionality. The functionality of the software is described in advance to discuss the added value with the engineers that will use

the software.

The method integrates concepts from existing domains of knowledge acquisition, modeling, automation and software development. The input and output of each step are standardized to allow a predictable development method. Standardization is done by using generic models of the design process and expert knowledge. Observations "how" designers design are used to define these models. The result is a generic procedure that starts with a design process and ends with software that generates multiple designs.

The first step of the method is to bring overview to the design environment. The original design context is divided, or decomposed, into distinct levels of abstraction, each with their own expressiveness and characteristics. The levels of abstraction discriminate between issues of higher or lower importance. A suitability check is provided to determine if the procedures from this thesis are applicable.

After the levels of abstraction are identified, the sub-process of analysis is used to prescribe the further breakdown into sub-processes and information. Analysis-oriented decomposition identifies three distinct types of information: performance, scenario and embodiment. The design process is divided in sub-processes of analysis, synthesis, evaluation and adjustment.

The decomposition phase is a key aspect for predictable and efficient modeling and software development. Decomposition allows *fast* knowledge acquisition, *less* complex modeling, automation with a *generic* and *predictable* software functionality.

The generic model of the design process is used to provide a standardized description of a design process. The functionalities of the software modules, as well as the complete system, are known at this point.

The step after decomposition acquires the expert knowledge and models it in a format called PaRC (acronym for Parameters, Resolve rule and Constrain rule). PaRC consists of entities to define the design artifact (parameters and topological elements) and knowledge rules that enable design generation (resolve, constrain and expand rules: R-, C- and X-rules). The acquired model describes the design expert's experience and know-how in solving design problems. The last steps of the procedure involve automation of the knowledge models and software development.

A generic software architecture mimics the model of the design process and has generic interfaces to the PaRC knowledge models. As a result, software development effort is reduced when building multiple software programs.

The proposed development method is applied to two industrial expert design cases and four cases with explicitly documented knowledge. The design process and expert knowledge are both modeled, and software prototypes are developed.

Samenvatting

Het productcreatieproces ondervindt een toenemende druk om producten van hoge kwaliteit in steeds kortere tijd te ontwikkelen. Informatie en Communicatie Technologie biedt een rijk scala aan oplossingen om de efficiëntie te verhogen en de concurrentie voor te blijven. Een veelbelovend type software ondersteunt het productcreatieproces door ontwerpalternatieven voor te stellen, op basis van gewenste product specificaties. Door automatisch vele alternatieve ontwerpen te genereren en deze aan de ontwerper te presenteren wordt een “oplossingsruimte” gecreëerd. Deze oplossingsruimte biedt in een vroeg stadium van het ontwerpproces inzicht in de mogelijkheden en beperkingen. Dit stelt de gebruiker in staat om met intuïtie en ervaring het beste ontwerp te kiezen voor een bepaalde toepassing. Ontwerpers besparen tijd, verhogen de kwaliteit van de uiteindelijke oplossing en verkrijgen hogere-orde kennis over de ontwerpproblematiek.

Dit proefschrift richt zich op software ondersteuning van ontwerpprocessen met expertkennis over bestaande technologieën, met parametrische informatie en kwantitatieve getalswaarden. Hierbinnen vallen continue en discontinue variabelen met een mix van lineaire en niet-lineaire vergelijkingen, logica en afschattingen, voor statische en dynamische topologieën. Het toepassingsgebied beslaat machine-elementen, productcomponenten en productsystemen.

De academische wereld heeft automatisering van vele typen ontwerpproblemen onderzocht, onder meer voor het toepassingsgebied van dit proefschrift. Een uitgebreide verzameling theorieën, modellen en technieken is ontwikkeld die geavanceerde ondersteuning mogelijk maken. Alhoewel de technische haalbaarheid lijkt te zijn aangetoond, worden automatisering van ontwerpprocessen niet veelvuldig in de industrie toegepast.

Dit proefschrift streeft naar hogere mate van ontwerpondersteuning voor de industrie. De focus ligt op het snel en efficiënt ontwikkelen van de modellen die noodzakelijk zijn om een ontwerpproces te kunnen automatiseren. De nadruk ligt op expertkennis voor de creatiefase van het ontwerpproces. Een methodische

aanpak wordt beschreven om de benodigde modellen te construeren en de software functionaliteit vast te stellen.

De methode is ontwikkeld door integratie van concepten uit bestaande onderzoeksgebieden als kennisacquisitie, modelvorming, automatisering en softwareontwikkeling. De gegevensuitwisseling tussen de diverse activiteiten is gestandaardiseerd om een voorspelbare procedure te documenteren. De standaardisatie maakt gebruik van generieke modellen van het ontwerpproces en expertkennis. De modellen zijn opgesteld aan de hand van observaties hoe een ontwerper ontwerpt en tot oplossingen komt. Het resultaat is een ontwikkelprocedure die het proces voorschrijft van (expert)ontwerper tot en met softwaresysteem.

De eerste stap is het in kaart brengen van de ontwerpcontext. Het ontwerpproces wordt onderverdeeld in abstractieniveaus met elk eigen informatie en processen. De abstractieniveaus verdelen het proces in zaken van hogere of lagere mate van belangrijkheid. Een test wijst uit of een abstractieniveau geschikt is voor automatisering op basis van methoden uit dit proefschrift.

Na identificatie van de abstractieniveaus, wordt deze verder verdeeld in processen en informatiesets. De analysemethode is hiervoor het centrale concept, waarbij drie typen informatie worden gedefinieerd: *performance*, *scenario* en *embodiment*. Het ontwerpproces wordt verder onderverdeeld in de processen *analyse*, *synthese*, *evaluatie* en *aanpassen*. Deze fase is de decompositie fase.

De decompositie fase is kritiek om voorspelbaar en efficiënt een softwareprogramma te kunnen ontwikkelen. Goede decompositie resulteert in snelle kennisacquisitie en minder complexe modellen die bovendien geautomatiseerd kunnen worden met een relatief simpel algoritme dat tevens generiek toepasbaar is. Hierdoor is de kernfunctionaliteit van de software in een vroeg stadium bekend.

De activiteit na decompositie is het verkrijgen en modelleren van de expertkennis in een beschrijving genaamd PaRC (acroniem voor Parameters, Resolve-regels en Constrain-regels). PaRC bestaat uit bouwstenen om een ontwerpobject te definiëren (parameters en topologische elementen) en kennisregels om ontwerpcreatie te simuleren (*resolve*-, *constrain*- en *expand*regels: R-, C- en X-regels).

Het kennisacquisitieproces begint met het eindresultaat van de decompositiefase. Specifieke vragen worden aangereikt om dit proces efficiënt te laten verlopen. Het verkregen model beschrijft de kennis en know-how om ontwerpen te creëren.

De laatste stap van de ontwikkelprocedure beschrijft automatisering van de kennismodellen en softwareontwikkeling. Een generieke softwarearchitectuur is gebaseerd op het model van het ontwerpproces en heeft generieke interfaces naar de PaRC kennismodellen. Hierdoor wordt de vereiste inspanning om meerdere softwaresystemen te ontwikkelen verder gereduceerd.

De beschreven ontwikkelprocedure voor ontwerpautomatisering is toegepast op twee industriële (expert)ontwerpproblemen en een viertal ontwerpproblemen die beschreven staan in handboeken. Modellen van het ontwerpproces en expertkennis zijn opgesteld, en prototype softwareprogramma's zijn ontwikkeld.

Table of Contents

Summary VII

Samenvatting IX

Table of Contents XI

1 Introduction 1

1.1 Software support for engineering design 1
1.2 Solution presentation 2
1.3 Multiple solutions 4
1.4 Software development 6
1.5 Focus and scope 7
1.6 Research hypothesis 7
1.7 Thesis outline 8

2 Literature 9

2.1 Theory of Technical Systems 9
2.2 General Design Theory 10
2.3 Function-Behavior-State 10
2.4 Knowledge Intensive Engineering Framework 12
2.5 KADS and KARL 13
2.6 Computational Synthesis 14
2.7 Algorithms 15
2.7.1 Constraint Programming 15
2.7.2 Optimization 15
2.8 MOKA 16
2.9 Knowledge Engineering 17
2.10 Previous research 18

3 Model of synthesis knowledge 23

3.1 A model of design 24
3.1.1 Information 24
3.1.2 Processes 25
3.1.3 Levels of abstraction 26

3.2 What is synthesis knowledge . . . 27
3.3 A model of synthesis knowledge . . . 29
3.3.1 Challenges . . . 29
3.3.2 Embodiment . . . 30
3.3.3 Knowledge rules . . . 31
3.4 Synthesis algorithm . . . 35
3.5 Limitations . . . 37
3.5.1 Systems of equations . . . 37
3.5.2 Consistency and solvability . . . 38
3.5.3 Revising decisions . . . 38
3.5.4 Algorithm . . . 39

4 Knowledge engineering method 41
4.1 Step 1: identify levels of abstraction . . . 43
4.2 Step 2: selection . . . 43
4.3 Step 3: analysis formalization . . . 44
4.3.1 Differences in analysis methods . . . 45
4.4 Step 4: synthesis formalization . . . 47
4.5 The knowledge document . . . 53
4.6 Limitations . . . 53

5 Software development method 55
5.1 Step 1: overview . . . 57
5.2 Step 2: selection . . . 57
5.3 Step 3: modeling . . . 57
5.4 Step 4: automation and implementation . . . 58
5.5 step 5: user interaction . . . 59

6 Implementation and realization 63
6.1 Architecture . . . 63
6.2 Industrial cases . . . 68
6.2.1 Optical chamber of an XRF spectrometer . . . 68
6.2.2 Baggage handling system . . . 77
6.3 Explicitly documented cases . . . 84
6.3.1 Belt drive . . . 84
6.3.2 Compression spring . . . 90
6.3.3 Extension spring . . . 92
6.3.4 Torsion spring . . . 93
6.4 Comparison . . . 94
6.4.1 R-rules . . . 94
6.4.2 Parameter dependency graphs . . . 96
6.4.3 Development time . . . 98

7 Conclusions & Recommendations 101

7.1 Conclusions . . . 101
7.2 Recommendations . . . 105
7.2.1 Industrial decomposition . . . 105
7.2.2 Knowledge acquisition . . . 105
7.2.3 Modeling and automation . . . 106
7.2.4 Generic software development . . . 106
7.2.5 User interaction . . . 107
7.2.6 General . . . 107

List of References **111**

Appendices

A Synthesis knowledge **119**
A.1 Optical chamber . . . 119
A.1.1 Elements and parameters . . . 119
A.1.2 X-rules . . . 121
A.1.3 R-rules . . . 121
A.1.4 C-rules . . . 122
A.2 Baggage handling systems . . . 123
A.2.1 Elements . . . 123
A.2.2 Parameters . . . 125
A.2.3 X-rules . . . 128
A.2.4 R-rules . . . 129
A.3 Compression spring . . . 130
A.3.1 Parameters . . . 130
A.3.2 R-rules . . . 131
A.3.3 C-rules . . . 132
A.4 Extension spring . . . 133
A.4.1 Parameters . . . 133
A.4.2 R-rules . . . 134
A.4.3 C-rules . . . 135
A.5 Torsion spring . . . 135
A.5.1 Parameters . . . 135
A.5.2 R-rules . . . 136
A.5.3 C-rules . . . 137

List of Figures

1.1 point, path and cloud solution spaces 2
1.2 software support for engineering design 3
1.3 online flight booking application (source: website KLM) 4
1.4 solutions for suspension design 5
1.5 knowledge domains and the software development 6

2.1 the FBPSS framework (after [56]) 11
2.2 the FBS framework (after [16]) 12
2.3 compression spring designer 20
2.4 synthesis module development time 21

3.1 embodiment and performance (image courtesy of COMSOL Inc) . 25
3.2 design process 26
3.3 levels of abstraction (after [26]) 27

4.1 knowledge engineering 42
4.2 three types of analysis 46
4.3 tube position 49
4.4 element types for functions 51
4.5 parameter dependency graph, example 52

5.1 software development 56
5.2 requirements input, belt drive case 60
5.3 solutions representation, belt drive case 61

6.1 architecture 64
6.2 central data storage, class model 65
6.3 synthesis module, class model 65
6.4 PaRC model, class model 65
6.5 implementation example 67
6.6 spectrometer and optical chamber (copyright PANalytical BV) . . 69
6.7 optical chamber, embodiment 72
6.8 parameter dependency graph, optical chamber 74
6.9 user interface, input 75

6.10 user interface, output . . . 75
6.12 baggage handling system (copyright Vanderlande Industries) . . . 77
6.13 process flow design (PFD) . . . 78
6.14 PFD process . . . 79
6.15 parameter dependency graph, baggage handling system . . . 82
6.16 material flow diagram . . . 83
6.17 belt drive, embodiment and scenario . . . 85
6.18 parameter dependency graph, belt drive . . . 88
6.19 graphical user interface, belt drive . . . 89
6.20 compression spring . . . 90
6.21 parameter dependency graph, compression spring . . . 91
6.22 extension spring . . . 92
6.23 torsion spring . . . 93
6.24 R-rules versus parameters . . . 94
6.25 number of R-rules per parameter . . . 95
6.26 comparison of parameter dependency graphs . . . 97
6.27 synthesis module development time . . . 99

7.1 knowledge domains and software development . . . 105
7.2 alternative development procedures . . . 108

A.1 optical axis . . . 121

Chapter 1

Introduction

We encounter in the world around us an enormous stream of products with constantly changing features and appearances. It seems that a product is (re)designed for nearly every taste, price range and user group. This trend of increasing product diversity has a profound impact on companies, teams and individuals that develop these products [33].

One of the reactions is to use existing technologies instead of innovative concepts to develop the required diversity of high quality products at affordable prices [18]. Development teams are supplied with a flexible network of internal and external technology sources to enable quick assimilation of existing technologies. However, this increases the time pressure on new product development because the same technologies are also available for the competition. Therefor, in order to remain competitive, one must increase one's product development efficiency.

Companies have several strategies to improve efficiency of development teams, one of which is implementation of software support for the design processes [36]. The research in this thesis aims to improve the software support for the (engineering) design processes that use existing technologies. The subsequent sections describe the currently available software and identify possible room for improvement.

1.1 Software support for engineering design

The fast development of consumer markets increases the pressure on the product design process to reduce time to market [21] [34]. Uncertainties and risks are reduced where possible [1] and information and communication technology is adopted to enhance flexibility, speed and efficiency of the process [36] [17].

A review of the commercially available software for (engineering) design reveals that the majority focuses on analysis, drawing and/or refining details of established design concepts [49] [42] [32]. Only little software support addresses the creation process of designs. The majority of software requires a fully defined design as input, which forces the engineer to plan ahead and make choices. After a weak point is identified by simulation, this can be corrected in a number of ways. Only little methodology is provided for guidance about what to do next.

Ullman [49] describes an ideal support system for the creation of new products: insight is provided in the relationship between the customer wishes and the available product options. The search for the best design is an automated process, guided by the preferences of the engineer. Support systems generate and present alternative solutions that give an overview of what is possible. This allows experts to use their experience and “design intent” to select the solution that is better than all others.

1.2 Solution presentation

A feature to distinguish support software for engineering design is the way each type represents a solution space, illustrates in Figure 1.1. A single software program can be of a single type or a mix of these types.

When an engineer finishes a design, he/she can execute analysis to reveal some quality characteristics, such as strength, dynamic response or a more intuitive judgment about aesthetics or user-friendliness. After analysis, a design is placed somewhere on a quality scale as a single point, Figure 1.1a. The point gives valuable information about the quality of a design, but reveals nothing about alternatives or what the limits are of achievable quality.

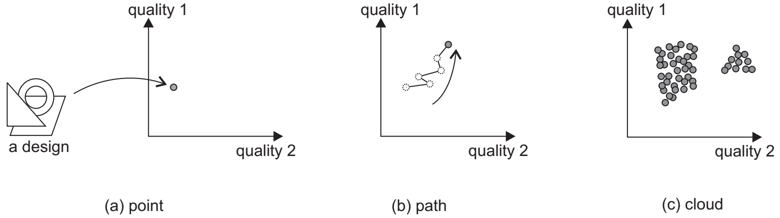
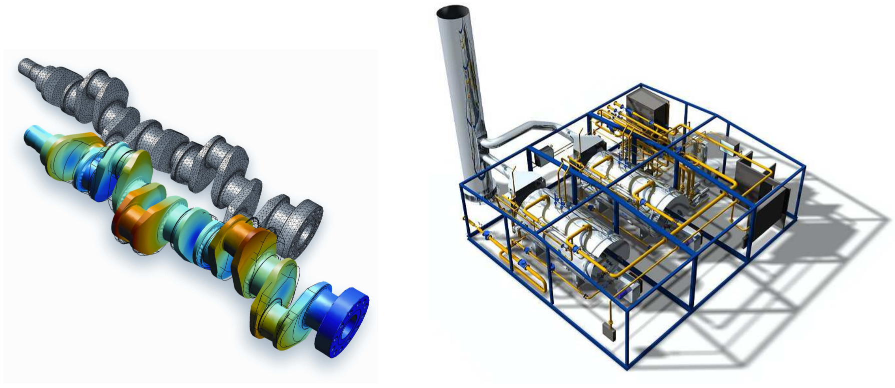


Figure 1.1: *point, path and cloud solution spaces*

Examples of software that gives a single point solution are shown in Figure 1.2: a finite element analysis of a crankshaft predicts its mechanical behavior before the part is produced, Figure 1.2a. The layout of a plant is analyzed to check if all pipes are designed correctly, Figure 1.2b. Analysis software is often essential to deliver high quality products, meet deadlines and prevent costly redesign.

The next type of software provides information about multiple designs: each design is made from a modification of a previous design, with the goal of opti-



(a) *finite element analysis of crankshaft (image courtesy of COMSOL Inc.)*

(b) *plant simulation (image courtesy of Vertex Systems Oy)*

Figure 1.2: *software support for engineering design*

mizing its quality, Figure 1.1b. The process requires a design as starting point and an objective function to navigate toward the optimum. An algorithm interprets the result of a certain action and decides what to do next. Optimization research is being done for decades, if not centuries. A wide range of techniques, literature and implemented toolboxes exist that can perform optimization on a mathematical model.

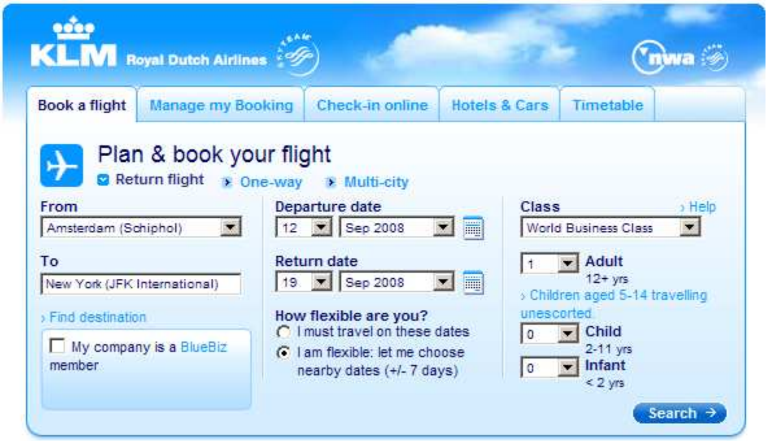
The third type of software provides a “cloud” of solution points that are not created based on other points, Figure 1.1c. The difference between the second and third type is that the third type contains only initial designs, where each point satisfies the modeling constraints. Each point of the cloud can serve as starting points for further optimization.

The intended support this thesis aims to provide to engineers is software that generates clouds. A cloud indicates the possibilities and limitations of design solutions, which is higher level design knowledge that is derived by observing the shape of a cloud. No explicit effort is taken (yet) to find the extremities of the solution space. Afterward, some solutions can be selected for further optimization, either by human or computational methods. In both cases, initial points are required for each new design problem and a cloud scattered across the solution space can give valuable information about global optima.

This thesis aims to develop software of the third type for engineering design. Multiple design solutions are generated based on a specification of the required product quality. The cloud-type software offers insight in the possibilities and limitations of engineering design solutions.

1.3 Multiple solutions

A simple (non-engineering) example of software that provides multiple solutions is an online flight booking applications, Figure 1.3. Consider the booking process of a flight from Amsterdam to New York. The software presents an overview of multiple alternatives to check and compare the prices and time schedules. The process to select the best option is an exploration of alternatives. The software accepts and supports the fact that the user knows more than the application. Its added value extends from pure automation of a task toward support for the entire process from customer wishes to solution: the software generates possible solutions and allows the user to select the best.



Plan & book your flight

Return flight · One-way · Multi-city

From: Amsterdam (Schiphol)

To: New York (JFK International)

☐ My company is a BlueBiz member

Departure date: 12 Sep 2008

Return date: 19 Sep 2008

How flexible are you?

○ I must travel on these dates

● I am flexible: let me choose nearby dates (+/- 7 days)

Class: World Business Class

1 Adult 12+ yrs

0 Child 2-11 yrs

0 Infant < 2 yrs

Figure 1.3: *online flight booking application (source: website KLM)*

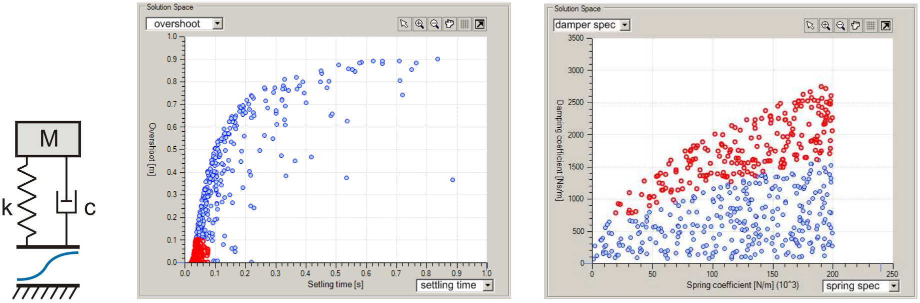
Software with multiple solutions for engineering design provides a more technically related view on the possibilities, limitations and qualities. An example is shown in Figure 1.4, for the design of a car suspension system that consists of a damper and a spring. The design goal is to comfortably absorb a bump in the road: the car should go up and down only slightly and quickly stop bouncing. Consider the situation of a car driving up a curb of say 10cm, at a speed of 5 $^{km}/_{h}$.

The designer has to find the right specifications for the suspension system, consisting of a spring and damper in parallel arrangement, indicated in Figure 1.4a. The software generates and analyzes multiple designs. The resulting solution space provides quantitative information about the possible behaviors of designs. The behaviors of interest are in this case the *height* and the *duration* of the bouncing motion. The software allows the designer to see what is possible, and select the solution he/she prefers.

Figure 1.4b depicts the quantitative behavior specifications of about 200 designs: the overshoot on the y-axis and the settling time on the x-axis. Each dot represents a quantified design that is generated. The best designs are located in the bottom-left corner (the red group).

Figure 1.4c shows a plot of the two main design variables: damping coefficient and spring coefficient. In this plot, the same group of best designs is located at the upper section of the solution cloud (the red group). Observing these two plots, the designer sees what springs and dampers will result in a comfortable ride.

The software from Figure 1.4 uses knowledge from commonly used engineering handbooks. However, the insight on the design solutions is difficult to obtain from theoretical analysis and trial-and-error iterations by hand. The software provides an overview of possibilities, which saves time and ensures an optimal design because the designer is aware of the alternatives and picks the design that suits him best.



(a) **(b)** *overshoot vs. settling time* **(c)** *design alternatives*

Figure 1.4: *solutions for suspension design*

The software discussed in this thesis propose solutions based on *knowledge rules*. A list of rules is defined beforehand and an algorithm operates upon this knowledge to find design solutions. Example of such knowledge rules are equations and if-then logic. Knowledge from experts is used when available and accepted as “truth”, even though the scientific rigor of such knowledge is perhaps not explicitly researched. However, there is a growing discrepancy between software support for consumers to reduce search time and improve decision quality (e.g. route planners and online shops) and engineering design. My goal is to enable large scale deployment of cloud-type design automation software for present-day engineering design.

Decades of academic research have explored many different approaches and algorithms to realize support systems with design automation for point, path and cloud-type solution representation. Theories, frameworks and algorithms have been developed to enable advanced forms of intelligent, knowledge-based support for increasingly complex problems. The technical feasibility for a large range of problems within engineering design has been proved.

A bottleneck that begins to emerge is the development process of the software systems themselves. For design problems with dynamic topologies, Cagan et al. [6] note this process is little documented. For optimization systems, Papalambros and Wilde [31] provide a checklist and guideline for computational optimization (discussed in more detail in Section 2.7.1). The modeling is done by someone who understands optimization and is able to translate the design case to a computational model and algorithm. Prototypes developed at my own university indicate that different developers lead to different models with different software functionality and require a different amount of time. My goal is to prescribe the modeling activity as specifically as possible, leading to a more controllable process and end result. The trade-off I will have is the applicable scope. Papalambros and Wilde [31] offer more general guidelines for a broader scope of problems, while I aim at specific development guidelines for a more narrow scope.

1.4 Software development

A number of concepts, methods and theories from several knowledge domains are required during development of design automation software, Figure 1.5. The continuous path through the domains, from left to right, indicates the development process of design automation software based on expert knowledge.

The development process begins with a knowledge source. The design processes of the source are first *decomposed* to reduce complexity, gain overview and determine suitable system boundaries for the software. After selection of a design process, the relevant knowledge is *acquired* and made verbally explicit. Next, the knowledge is *modeled* into a format that is subsequently *automated* by an algorithm. When developing several different software applications, the concept of *generic software development* is used to reduce the required effort. Finally, the *user interaction* is determined to offer the best interaction and highest added value for the end-user.

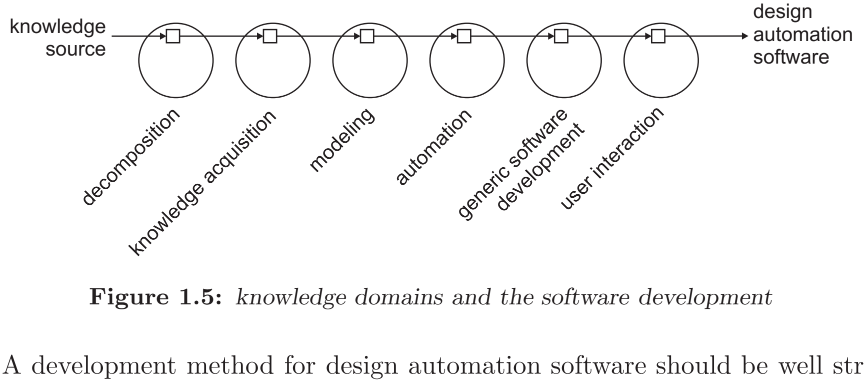


Figure 1.5: *knowledge domains and the software development*

A development method for design automation software should be well structured and documented, and have a clear applicability scope. The person to execute the method is not forced to become an expert in all the knowledge domains, but is guided by a continuous procedure. This includes the activities from first meeting with the experts, to implementation of the knowledge base.

This thesis proposes a development method for design automation applications and uses a model of synthesis knowledge as leading concept to integrate the knowledge domains.

1.5 Focus and scope

Software that creates many initial solutions requires, in the minimum, a module that *creates* a design and a module that *analyzes* a design. The act of design creation is labeled ”synthesis”. This term is chosen to emphasize its opposition with analysis: synthesis begins with required specifications and results in a design, while analysis begins with a design and results in quality information. Synthesis knowledge is all information and relations that are used to generate a design. The knowledge engineering activities to obtain the model of synthesis knowledge is the focus of this thesis.

The scope of this thesis is engineering design that uses existing knowledge of known technologies. The knowledge itself has parametric information and quantitative data. The source of this knowledge is available, either as human expert designer or explicitly documented.

1.6 Research hypothesis

The hypothesis of this thesis is:

a model of synthesis knowledge forms the basis of a knowledge engineering method for the development of design automation software.

With the following definitions:

- model: a simplified mathematical description of a system or process, used to assist calculations and predictions;
- knowledge engineering: the process to develop or design a computational model of knowledge;
- synthesis knowledge: information and relations acquired through experience or education that are used during the synthesis phase of design: where designs are generated;
- design automation software: software that generates (multiple) designs.

The hypothesis is tested within the previously described scope.

1.7 Thesis outline

A literature survey is made to position the scope and research goal relative to existing nomenclature and research projects. Section 3 proposes a model of the design process that is generic within scope and aimed at development of design automation software. The input and output of the synthesis phase is defined and a model for the activity of synthesis proposed. A knowledge engineering method is derived from this synthesis model to acquire and model the relevant knowledge from literature or human source, discussed in Section 4. How this method is used during software development is explained in Section 5. Chapter 6 describes the implementation of the development method and knowledge engineering method for two cases from industry, and four cases from engineering handbooks. Prototypes are developed and the models of synthesis knowledge are compared to each other. Chapter 7 concludes the thesis and proposes several future directions of research.

Chapter 2

Literature

An overview of several theories and research programs is provided to position the research of this thesis.

First, the Theory of Technical Systems, General Design Theory and the Function-Behavior-State model is addressed for general referencing. Next, the focus shifts to projects of computational design support such as the framework of the Knowledge Intensive Engineering Framework (KIEF), the KADS research project and its more formal language KARL. Subsequently, Computational Synthesis is discussed together with algorithms of Constraint Programming and optimization. The development process of design support systems is discussed by the MOKA project and one section is dedicated to knowledge engineering. The research history at the Department of Design, Production and Management of the University of Twente is described to illustrate the context in which this research is conducted.

2.1 Theory of Technical Systems

The Theory of Technical Systems (TTS) [19] explores the design process as broad as possible, and aims to organize, store and reference all knowledge for and about design. Central to the theory is the engineering design process to design a Technical System (TS). A TS is a transformation system that is described in processes, functions, organs and components.

The lowest levels of detail are the so-called properties. These properties are all those features which belong substantially to the object. Two types of properties exist: internal and external. Internal properties are under the control of the engineering designer, such as the structure description (components, arrangements), forms and dimension. The external properties are the observable and detectable properties. In general, TTS gives a broad, qualitative description on the relationships between properties.

The knowledge of TTS relates the design process at enterprise level, but also descends in level of detail to *technical knowledge*: knowledge about artificial objects which have been created and produced to accomplish certain goals. TTS identifies several kinds of knowledge, such as basic knowledge about strength, materials, manufacturing, and functional knowledge about models and processes. Knowledge is available explicitly or tacitly and should always bring an answer to an immediate question.

This thesis focuses on the “property” entities of TSS and the knowledge rules that relate them.

2.2 General Design Theory

The General Design Theory (GDT) is a formal theory of design knowledge to clarify the human ability of designing in a scientific way [54]. It also aims to produce practical knowledge about design methodology and the construction of CAD systems [54].

GDT describes the design of artifacts to fulfill functions. The inputs of the design process are specifications, and design itself is a process of mapping these specifications within a functional space to an attribute space. Design is a stepwise refinement process mediated by metamodels, toward a definite description of the design object in the attribute space. The attribute space is the definition of the design object with sufficient level of detail to be manufactured.

The theory of GDT describes design in a broad sense, while this thesis considers parametric design of existing entities. In GDT terms, the software developed from this thesis aims to explore the neighborhood of an entity in the attribute space. Due to discontinuous degrees of freedom of the entity this might not be a continuous space, but still relatively predictable.

2.3 Function-Behavior-State

The Function-Behavior-State concepts offer a language to describe design object in different dimensions [50] [56]. To position this thesis, I discuss the concept of structure as well, to explicitly differentiate between the design artifact and changes in its attributes.

Figure 2.1 depicts the dimensions of Function, Behavior, State and Structure relative to each other. The “lowest” dimension is the *structure* of the design artifact, i.e. what it is. Putting an artifact to work is described in the dimension *state*. The states can be series of structure attributes, flows of information, material and energy. The overall relationship between these states describes the *behavior* of the artifact, i.e. what it does. Finally, at the highest dimension, the behavior is used to fulfill a certain *function* within a larger context.

Beside the dimensions is the concept of *principle*, which governs the fundamental relations between structure, state and behavior. These are the laws of e.g.

physics or kinematics that allow development of quantitative relations between the different dimensions.

This thesis focuses on the structure and state dimensions, with explicitly known relations to behavior.

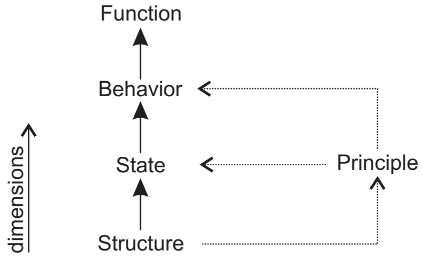


Figure 2.1: *the FBPSS framework (after [56])*

The *process* of designing is modeled by Gero and Kannengieser using the situated Function-Behavior-Structure (sFBS) framework [16]. Here, slightly different definitions of the concepts are used. In short, the *function* describes what the object is for, *behavior* describes what it does and the *structure* describes what it is.

A brief description of the processes of the sFBS framework is given, as depicted in Figure 2.2 (solid lines are processes also described in this thesis): the design process begins by the formulation process (1). This translates the required function into behavior that is expected to enable this function. A synthesis process (2) generates a structure based on this expected behavior. Once a candidate structure is generated, process (3) analyzes this to derive its actual behavior. The evaluation process (4) compares the actual behavior with the expected, and decides the next step. Process (5) produces the documentation of the structure for constructing or manufacturing the product. After evaluation, three types of reformulation processes are possible: process (6) addresses changes in the structure description; process (7) addresses changes in the behavior variables and process (8) does this in the functional variables.

The sFBS framework of the design process further consists of three interactive worlds:

1. external world: the world composed of representations outside the designer or design agent;

2. interpreted world: the world that is built up inside the designer or design agent. This world is seen as an abstraction of the external world;

3. expected world: the world that the imagined actions of the designer or design agent will produce.

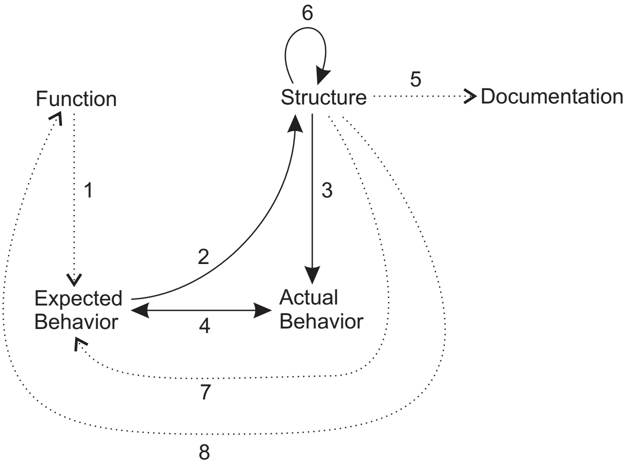


Figure 2.2: *the FBS framework (after [16])*

This thesis models a quantitative, parametric design process as an interpreted world representation for specific, quantifiable behavior variables. Knowledge rules are divided into the processes of synthesis, analysis, evaluation and adjustment. An explicit division is made between the artifact model and the knowledge rules that govern the relations between them. The link to function and documentation is outside the scope.

2.4 Knowledge Intensive Engineering Framework

The Knowledge Intensive Engineering Framework (KIEF) supports the design process by means of a software assistant that predicts a design object's behaviors across domains [55]. KIEF integrates domain knowledge such as electronics and dynamics, allowing multi-domain analysis, causality of phenomena and qualitative reasoning about behavior.

The building blocks of KIEF are related to an ontology of physical concepts, much like geometric features. The knowledge of domain theories is related to these physical concepts, which are stored in a library to allow for a faster and more expressive support system. Modeling design objects in multiple domains is done through a "metamodel". This metamodel exists on the abstraction level above the domains and enables flexible integration of their knowledge theories.

First, a description of a new design object is made in different domains and connected to the metamodel. An initial metamodel describes the design object in conceptual and topological terms. The second step involves enriching the initial metamodel with causality knowledge to a point where the design solutions can be reasoned upon. The last step is the actual use of KIEF as a design assistant: the prediction of unexpected physical phenomena across domains.

In relation to KIEF, the content of this thesis is domain agnostic. KIEF appears to focus more on the qualitative and/or causal analysis and simulation of design objects on the behavior level, while this thesis relates to the generation of design objects themselves, and the knowledge required for this.

2.5 KADS and KARL

The Knowledge Acquisition and Documentation Structuring (KADS) research offers a structured development process for knowledge-based systems [53]. It focuses on Expert Systems (ES) that reason about situations with the goal of extending the situation description to reveal causality, i.e. what happens and *why*. In order to do so, the ES requires a knowledge base that describes facts, conditions, inferences and dependencies. This causality knowledge is challenging to acquire from domain experts, as they are experts in problem solving, not in *explaining* their solutions [14]. As a result, development methods for ES moved away from the concept of knowledge acquisition as “direct knowledge transfer” and instead introduce a Knowledge Engineering (KE) process.

During this KE process, a specialized *knowledge engineer* develops, or designs, a computational model of some expert’s knowledge. This cyclic process requires the knowledge engineer to observe and interpret the original knowledge, and verify the correctness of the new computational model. KADS offers several semi-formal models to structure the knowledge of experts and aid the modeling activity. A further formalization of this approach is the Knowledge Acquisition and Representation Language (KARL) [14]. KARL supports the process to formalize the knowledge from knowledge engineer into a software language. The result is a formal modeling language that can infer and reason without supervision, given certain strict mathematical conditions. Because of the formalization, KARL provides support such as graphical representation and an interpreter and debugger of knowledge. One advantage of this knowledge engineering approach is that the knowledge model can have high expressiveness and the problem solving capabilities exceed that of a single expert.

Summarizing, one could say that knowledge-based systems that reason about cause and effect require knowledge engineering because the causal knowledge is difficult to extract directly. The modeling is done in a formal modeling language because the reasoning algorithm require mathematical rigor to reason through the knowledge base autonomously.

The type of engineering design problems addressed in this thesis do not require causality knowledge, because only automation is required. This reduces the need for knowledge acquisition of causality knowledge and a formal mathematical language or reasoning algorithm.

2.6 Computational Synthesis

A research overview of automation and optimization of design problems with variable topologies is given by Chakrabarti [9] and by Antonsson and Cagan [2].

The A-Design theory to computational synthesis implements an agent-based approach [8]. Four classes of goal-directed agents are used to generate a wide range of solutions. Configuration-agents create solutions qualitatively by random selections of component and connect their input with output. Instantiation-agents fix component values, determining the parameters of the design. Modification of existing solutions is done by the fragmentation-agents. Each agent is given a preference while performing its task, resulting in a broad exploration of the solution space. User preferences and learning algorithms from past designs are used to influence the solution generation process through manager-agents. These agents steer the optimization and search process by adjusting the goals of the other agents.

Computational synthesis using the A-Design theory offers support for design processes ranging from shape driven (architectural) design [8] [2] to electro-mechanical systems [7]. It has also proved efficient in the travelling salesperson problem and allows self-learning, as presented by Moss et al. [29]. A generic flowchart for computational synthesis has emerged for agent-based synthesis tools [6].

The concept of grammars offers a formalization of design synthesis knowledge. The result is a form of production rules, or graph based pattern recognitions that expand an initial graph into an eventual design. It supports geometric representations, reasoning and emergent shape properties [2].

Within the mechanical engineering domain, a grammar is a mapping between a function of an artifact and its form [23]. Generating solutions on both the topological and parametric level can be done using so-called parallel grammars, e.g. for gear design [44].

Graph grammars are used for topologies, networks of elements and to represent conceptual functions of a design, e.g. [43] and [23], but also neural networks [51]. Because graph grammars modify a valid graph into another valid graph, each state can be analyzed. This enables simultaneous synthesis and optimization in e.g. MEMS design [5].

The class of design problems addressed in this thesis also have topological degrees of freedom, but can be described parametrically. This thesis further focuses on the generation of *initial* designs. Until that initial design is found, no analysis is possible.

2.7 Algorithms

A wide range of algorithms is developed for the scope of problems I address. The two groups of algorithms discussed briefly here provide a view of the wide range of algorithms that can be applied once a model is defined.

2.7.1 Constraint Programming

Barták [3] provides an overview of the solving technology of Constraint Programming to automate design processes and generate multiple solutions. Constraint Programming is a method of problem solving that allows declarative specifications of relations among objects.

For the generation of initial solutions, as intended in this thesis, the constraint *satisfaction* algorithms are especially relevant, as systematic or stochastic search. Constraint satisfaction generates initial solutions that satisfy the constraints, without further solution modification or optimization. Examples are generate-and-test, backtracking (incremental expansion), the group of consistency techniques and constraint propagation [22].

The backtracking algorithm is a basic but robust algorithm that is likely to find one or more solutions and is interesting to use as a baseline.

2.7.2 Optimization

Many engineering problems require not just any good solution, but the optimal solution. Optimization algorithms generate solutions toward an optimum, defined by an objective function. Marler and Arora [24] present a survey of continuous nonlinear multi-objective optimization methods for engineering problems.

Especially relevant for software that generates multiple solutions are the Pareto optimal points: solutions that lie on the boundary of the solution space, and cannot be improved in one performance without deteriorating in another.

One of the most common methods to handle multi-objective optimization is to combine all objective functions into a single global function. Adding weights to each individual objective allows modeling of engineering preferences. A different approach is the "bounded objective function method", which offers a hierarchy in objective functions to separate between mandatory and additional objectives that are to be minimized.

Avoiding local optimality is important to offer a higher level interpretation of the solution space. Methods such as Tabu-search offer a heuristic procedure for solving optimization problems, designed to guide other methods to escape the trap of local optimality. Simulated Annealing [20] offers a stochastic optimization procedure to find global optima and is widely applied in optimization problems.

The scope of problems addressed in this thesis has mixed continuous and discontinuous variables and non-linear relations of algebraic formulas and with logic. Because I aim to generate a sufficiently filled solution space, a certain amount of random walk in the algorithm is required. In a later stage, the algorithm can navigate more intelligently to find the Pareto solutions, thus giving the cloud clear outlines. Exploration of the solution space would benefit from optimization methods to handle multi-objective optimization.

A wide range of optimization algorithms are available in literature and implemented for engineering problems. All these algorithms require a model to operate upon, and it is the goal of this thesis to supply the models.

2.8 MOKA

MOKA is a European research project that started in 1998 and was active for 30 months. It provides a methodology to develop Knowledge-Based Engineering (KBE) applications. MOKA is an acronym for Methodology and software tools Oriented to Knowledge based engineering Applications. The goal is to reduce the investment and risk of KBE development: similar to this thesis. The scope is routine design in engineering with a strong link to geometry [27].

The MOKA approach prescribes the knowledge engineering process and supports it with a software tool. A standardization of knowledge was developed, called ICARE (acronym for Illustration, Constraint, Activities, Rules and Entities). ICARE is divided into a part that describes the design object (constraints, entities and illustrations) and a part to describe the design process (illustrations, activities and rules). The entire process of KBE development is described as follows:

1. knowledge gathering: collection of raw knowledge from design experts. A broad view on the design object, processes, related aspects and background information;

2. structuring: develop the so-called “Informal” model of knowledge, divided into object information and design process descriptions. The ICARE concepts are used to facilitate this step and the next;

3. formalizing: refine the Informal model and develop a rigorous “Formal” model of the application knowledge, that is used to build the KBE system. This model consists of two sections: the “Product Model” that describes the object and related knowledge, and the “Design Process Model” defines the execution and decision making order, plus the process of selection choices;

4. implementation: software development of a KBE application.

MOKA focuses on the second and third step. Software tools are developed to allow non-KBE specialists to structure and formalize the relevant knowledge using the ICARE concepts. The process is methodologically described in the MOKA handbook [27].

Several commonalities and differences are identified between MOKA and this thesis. The goals are quite similar: reduce the development effort of knowledge-based software to support the design process. But, there are also some differences. MOKA has a strong link to geometry and geometric modeling: it uses assemblies and parts explicitly. This thesis does not do this, instead it adopts concepts of parameters and topological elements. MOKA does not prescribe the knowledge gathering step to determine the system boundaries and acquire the relevant knowledge. This thesis aims to do so.

MOKA's scope is wider compared to this thesis: aiming at any engineering design knowledge. Perhaps due to this wide scope, MOKA handles the solving algorithm (the Design Process Model) as case specific knowledge. This thesis has a narrower scope but uses a generic solving algorithm.

2.9 Knowledge Engineering

Knowledge Engineering (KE) is the process to design or develop a computational model of knowledge. KE involves activities of knowledge acquisition and representation [12] [13], both processes that have some distinct challenges.

Knowledge acquisition is the step during KE where design knowledge is made explicit. Schilstra [38] gives an overview of the development process of Expert Systems, and identifies several bottlenecks still persisting. These also include the tacit nature of expert knowledge, the challenges of knowledge extraction and the difficulty in modeling or representing the rules. In short, the well-known knowledge acquisition bottleneck [13].

Fensel [14] describes one of these difficulties by observing that design experts are experts in problem solving, not in explaining their solutions. The knowledge engineer therefore has to obtain thorough understanding of the problem at hand. Indeed, the book "Fundamentals of Computer Aided-Engineering" by Raphael and Smith states that the most successful engineering knowledge systems have been created for situations where the engineer-developers were also well acquainted with the subject [35].

In general, KE is seen as an activity that requires understanding of both the computational aspects as well as the design case at hand. The knowledge engineer has choices to make during the representation of knowledge into rules. For the design of shape grammars, for instance, the ideal grammar should be comprehensive yet model only feasible designs [2]. This involves choices regarding the amount of rules: many relatively simple ones, or a single complex rule? And the level of parametrization of the problem: many parameters for good expressiveness, or fewer for better computational performance? And how to describe the dependencies that occur between the rules?

The KE activity is usually done by people who *possess* the knowledge. This trend is seen in other research projects as well. The Knowledge Acquisition and Representation Language (KARL [14]) is aimed at knowledge acquisition from the *knowledge engineer* into a formal language. The MOKA project addresses the issues how to standardize and model design knowledge consistently, once it is gathered [27].

Studer et al. [45] reviews the principles and methods of knowledge engineering research. The modeling activity of expert knowledge includes the process of acquiring tacit knowledge and make this explicit. When re-usable problem solving methods are used, the process of knowledge modeling is prescribed using the generic roles that knowledge can play. This “shell” approach is used for parametric design tasks. However, the inflexibility of the problem solving method and the connection to the real-life situations remain a challenge. A proposal is to make a more flexible, configurable set of problem solvers.

This thesis aims to prescribe the KE activities from design process decomposition, knowledge acquisition, modeling and implementation. The goal is not only to make this process more predictable, but also for non-experts to be able to execute it. A model of synthesis knowledge is used as leading concept to select, optimize and integrate the most appropriate ingredients from existing domains of research.

The KE process addresses three major questions:

1. what is relevant: what are the system boundaries;
2. how to acquire the computational model of (synthesis) knowledge (knowledge acquisition);
3. how to automate the computational model: a generative algorithm.

The methods from this thesis aims to answer these three questions, without the knowledge engineer becoming a design expert him/herself. Ideally, the answers are stated by the source of the knowledge, during knowledge acquisition. The answers of the expert are implemented directly, with as little intermediate translation or modeling by the knowledge engineer as possible.

The modeled knowledge forms the *conclusion* of design experts: after years of experience it is finally known what is important and how to generate solutions efficiently. Using the proposed method, the knowledge is acquired and made explicit for the organization.

2.10 Previous research

Research projects at the Laboratory of Design, Production and Management of the University of Twente has been focused on intelligent design support tools for decades. An example is the FROOM project (acronym for Features and Relations used in Object Oriented Modeling) that supports the process of re-design, taking into account the manufacturing and process planning aspects of design decisions [37]. Interactive features with associated knowledge are used to allow definition and manipulation of geometry on higher levels than 3D drawing. The designer is informed of the consequences of design decisions in an early stage of design. Such functionality increases design efficiency and results in higher quality designs. The research project, of which this thesis is a part, further develops the view of supporting engineers to "look ahead" to see consequences of their choices and the limitations of solution spaces.

Recent research projects explore the use of Virtual Reality (VR) during the design process to include the end-user in the process. Tideman [46] proposed a method that uses scenarios, VR simulations and gaming principles to support designers. His method allows the different stakeholders of a design process (such as end-users, marketing managers, maintenance specialists) to create their own design and immediately test these in a wide range of scenarios. This proactive role of the stakeholders aids the designer during the design process.

The added value of VR technology is further explored in the "Synthetic Environments" research project (synthetic with the connotation of "artificial" or "man-made"). This project aims to provide a virtual prototyping environment for designers to see and feel (through haptic feedback) the implications of design decisions. The question how to efficiently develop such an environment for a design problem is among the core issues of a research project that started in 2005 [28]. My research is similar in the sense that both projects aim to bring a certain technology to industry.

This thesis is part of the research project "Smart Synthesis Tools" that started in 2005. The aim of this project is to provide engineering aid through automatic generation of design solutions. One of the first tools to explore this type of support is the WATT software for mechanism design [10]. The paper by Draijer and Kokkeler discusses the seed of the philosophy that led to the "Smart Synthesis Tools" (SST) project.

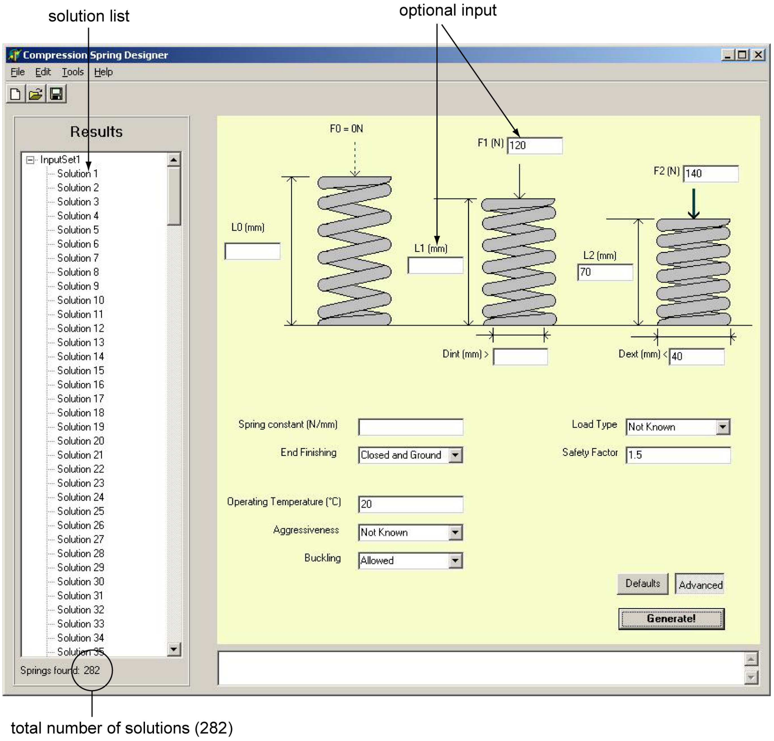
Research topics of the SST project address problem structuring, mathematical techniques, qualitative relations and handling of expert knowledge (this thesis). More details concerning the SST project are found in [41].

The question that sparked the research is related to the process of design automation development: it is an unpredictable endeavor and the required effort is not always in relation to its added value. To conclude this statement scientifically requires quantification of the development effort for a sufficiently large number of design problems, in a controlled environment.

Although a number of prototypes are developed at the University of Twente [40], the scientific rigor is insufficient to draw any firm conclusions. However, we can use these prototypes to illustrate (in a non-scientific manner) the unpredictable nature of the development process. I estimate the development time for the software module that performs synthesis, including knowledge acquisition, algorithm design, implementation and testing. The complexity of the synthesis module is measured by the number of parameters that a user can optionally specify as input: the degrees of freedom. The synthesis module has to cope with changing input specifications and generate solutions that satisfy this input. For the data points I assume an error margin of 20 to 30%.

First, the functionality of six prototypes is given, as well as the development time of the synthesis module (not the complete application). After that, the development time is plotted against the number of degrees of freedom to illustrate the apparent lack of correlation.

One of the first prototypes is developed for compression spring design, with the graphical user interface as depicted in Figure 2.3. The interface shows the spring in three positions: relaxed, in first compression and second compression mode. Several input fields are visible where the user can optionally specify several geometric requirements and spring characteristics, in this case values for F1, F2, L2 and a maximum value for the external diameter. The synthesis algorithm generates a list of fully defined springs that meet the specifications, in this case a total number of 282 springs. Each solution is parametrically fully defined in terms of material, geometry and usage situation. Knowledge is used from an engineering handbook [25] and DIN standards. Development of the code that performs synthesis took roughly 12-16 weeks.



solution list

optional input

Compression Spring Designer

Results

InputSet1
Solution 1
Solution 2
Solution 3
Solution 4
Solution 5
Solution 6
Solution 7
Solution 8
Solution 9
Solution 10
Solution 11
Solution 12
Solution 13
Solution 14
Solution 15
Solution 16
Solution 17
Solution 18
Solution 19
Solution 20
Solution 21
Solution 22
Solution 23
Solution 24
Solution 25
Solution 26
Solution 27
Solution 28
Solution 29
Solution 30
Solution 31
Solution 32
Solution 33
Solution 34
Solution 35

Springs found: 282

F0 = 0N
F1 (N) 120
F2 (N) 140
L0 (mm)
L1 (mm)
L2 (mm) 70
Dint (mm) >
Dext (mm) < 40

Spring constant (N/mm)
End Finishing: Closed and Ground
Operating Temperature (°C): 20
Aggressiveness: Not Known
Buckling: Allowed
Load Type: Not Known
Safety Factor: 1.5

total number of solutions (282)

Figure 2.3: *compression spring designer*

Prototypes with similar functionality as the compression spring designer are developed for spindle-drives, three types of springs and fiber-reinforced composites. The functionality of these prototypes is discussed briefly.

The spindle-drive designer enables the user to specify a desired motion and dynamic behavior of a manipulator that is positioned using a spindle-drive, powered by an electro motor and gear transmission. The software generates several combinations of electro motor, transmission and spindle. The synthesis algorithm takes into account the domains of dynamics, electronics and control systems and took approximately 14-18 weeks to develop.

Three spring designers are developed after the first version of Figure 2.3, for compression, extension and torsion springs. The development effort of the three systems was reducing due to the learning effect and re-use of code: approximately 10-14 weeks, 4-6 weeks and 3-5 weeks respectively.

The last prototype is the composite designer, which supports the design of fiber-reinforced composites. The user can specify which materials are allowed and information about the orientation and stacking of the plies. Different synthesis and optimization algorithms were implemented but the first algorithm (full search) took approximately 2 weeks.

The relation between the development time and number of degrees of freedom is given in Figure 2.4. The figure illustrates the unpredictable nature of the development of the synthesis module: no apparent correlation exists between the development time and the complexity of the problem. Secondly, a development time of several months *only* for the design generation module of a mechanical spring seems out of proportion.

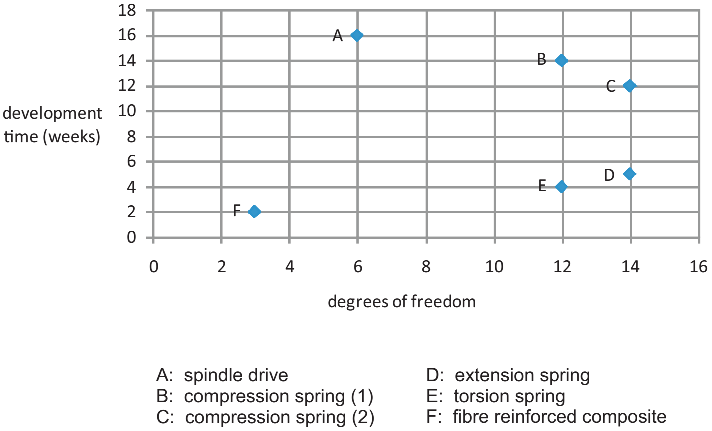


Figure 2.4: *synthesis module development time*

Chapter 3

Model of synthesis knowledge

The hypothesis of this thesis speaks of a model of synthesis that is ultimately intended to be automated. Such automation will simulate an *act* of synthesis. This chapter proposes a model of synthesis to describe this activity. The model consists of three parts: a description of the design artifact, a set of knowledge rules and an algorithm to combine these two and perform the act of synthesis.

From a cognitive point of view, synthesis can be seen as a construction of representations [52]. The synthesis phase starts with the design requirements and elaborates this representation toward the description of a complete design. The role of knowledge during this activity is not completely clear, but Wittgenstein used the phrase:

> *"To know means to know how to go on"* (Ludwig Wittgenstein)

An interpretation of Wittgenstein's view on knowledge is that it enables the owner to move from some begin to some end. This chapter proposes a description of synthesis knowledge, based on the view that knowledge enables the owner to take the next step.

First, a model of the design process is provided to position the activity of synthesis within the larger context, after which a more in depth discussion on synthesis follows.

3.1 A model of design

A design process consists of a series of processes that starts with the requirements and ends with a specification of a design. This section models a design process, its sub-processes and sub-sets of information. First, the sets of information are introduced, after which the flow of this information through the sub-processes is given.

3.1.1 Information

The goal of a design process is to arrive at a description of a "design" that satisfies certain requirements. Such a design does not necessarily describe geometry or a physical product, but can also be a layout or sketch. The concept of *embodiment* is introduced as the representation of the design artifact or system that is being designed. An embodiment can be a physical product such as a spring or machine, but also a control system or network layout. Higher level abstractions of springs or machines are represented by fewer parameters, but are still embodiments.

Embodiments are not designed to sit on a shelf doing nothing, but to be used in a certain situation. The description of such a usage situation is called a *scenario*. The scenario is information a designer cannot freely change. For instance, a spring is being deformed and a machine is switched on. A network layout is subjected to one or more scenarios, such as set of input signals. In case of the design of a construction beam, a scenario is the load that it has to support, or the temperature it has to withstand. The designer can change the design of the beam, but not the force that it will receive. The scenario is typically dictated from outside the design process, such a customer or previous design process.

After the embodiment and scenario are both known, the quality of the embodiment in that scenario is determined. The *performance* is the behavior that an embodiment exhibits in a scenario. For example, a construction beam has internal material stresses due to the load being applied, and a machine has certain dynamic performances such as overshoot and error margins.

Figure 3.1 shows an example of an embodiment and performance. The embodiment is a geometric model of a rim. The scenario is a fixed displacement and the performance is the resulting strain.

Because this thesis focuses on quantitative parametric designs, the embodiment, scenario and performance are all described in quantifiable parameters: information entities that receive a value during the design process. Parameters can be of different types, such as discrete, continuous, integers, predicates and sets (e.g. a material with a collection of properties).

In addition to the before mentioned independent types of parameters, a fourth type is introduced to express extra (dependent) information such as design intent and temporary construction parameters: *auxiliary* information. Examples are ratios, temporary estimates and the number of parts of an assembly. A model

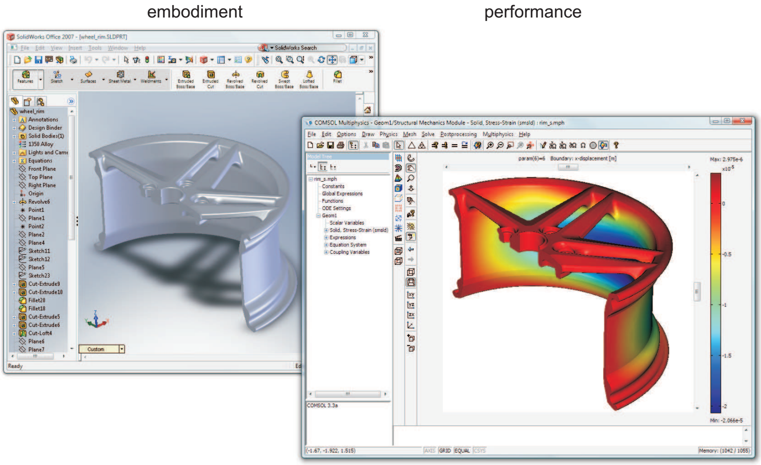

embodiment
performance
SolidWorks Office 2007 - [wheel_rim.SLDPRT]
COMSOL Multiphysics - Geom1/Structural Mechanics Module - Solid, Stress-Strain (smsld) : rim_s.mph

Figure 3.1: *embodiment and performance (image courtesy of COMSOL Inc)*

can be expanded with a great number of auxiliary parameters, but it is still the same embodiment.

At the beginning of the design process, a designer is likely to have certain *requirement* information concerning certain parameter values. The goal of the design process is to obtain an embodiment that meets these requirements. Requirements can relate to the embodiment, scenario, auxiliary and performance information. Examples of embodiment requirements are size and material restrictions. Performance requirements state maximum stress peaks or some dynamic system responses. These can be fixed numbers, maximum or minimum limits or general optimization goals.

The processes to move from requirements to solutions are discussed in the following section.

3.1.2 Processes

The design process begins with requirements and ends with a solution, through a number of smaller processes as modeled in Figure 3.2. The first process is where an initial embodiment is generated based on the requirements. This phase is termed “synthesis”. The embodiment description is with sufficient level of detail to enable analysis, which is the next process: analysis determines the performance of an embodiment in a given scenario. Analysis can only be executed after an embodiment is known, and a scenario is given. Analysis methods can vary from rules of thumb, formulas, Microsoft Excel-sheets, finite element calculations to dedicated simulation software.

The name synthesis is used to emphasize its opposition to analysis: synthesis begins with required behavior and ends with a design, while analysis begins with a design and ends with a specification on behavior.

After analysis comes the evaluation. This process compares the analyzed performance with the requirements and decides what to do with the embodiment next. Three options exist:

1. an embodiment seems promising, an adjustment is made to it, after which it re-enters analysis. An automated loop of analysis, evaluation and adjustment results in an optimization process. The optimization method coordinates the embodiment adjustments to steer toward the optimum;

2. an embodiment does not meet the requirements, nor is it expected to. It is abandoned and synthesis is initiated again;

3. the requirements are met and the embodiment is added to the solution list.

Improving an embodiment involves an adjustment process to modify the embodiment with the goal to better meet the requirements. The difference with synthesis is that both the embodiment and its performance are known.

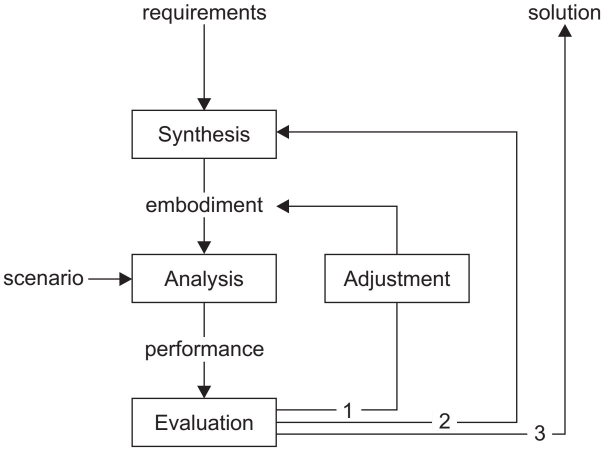


Figure 3.2: *design process*

The synthesis process is considered a core activity in the design process [11] [47]. Section 3.3 proposes a more detailed description of synthesis knowledge and Section 3.4 discusses an algorithm for automation.

3.1.3 Levels of abstraction

A design artifact can be represented by embodiments with a lower or higher expressiveness, known as levels of abstraction [26]. Descriptions with higher expressiveness take more details into account, and allow for more accurate analysis.

Levels of abstractions are seen as levels of expressiveness that are required to determine performance identifiers.

As an example, Figure 3.3 shows a machine component and a compression spring on a higher and lower level of abstraction. On the higher level, the representations contain fewer expressiveness, while the lower level has is more information. The analysis methods of each level take effects into account that allow for different, more or less accurate analysis. Performance information with high importance is likely to be checked on higher levels of abstraction, early in the design process.

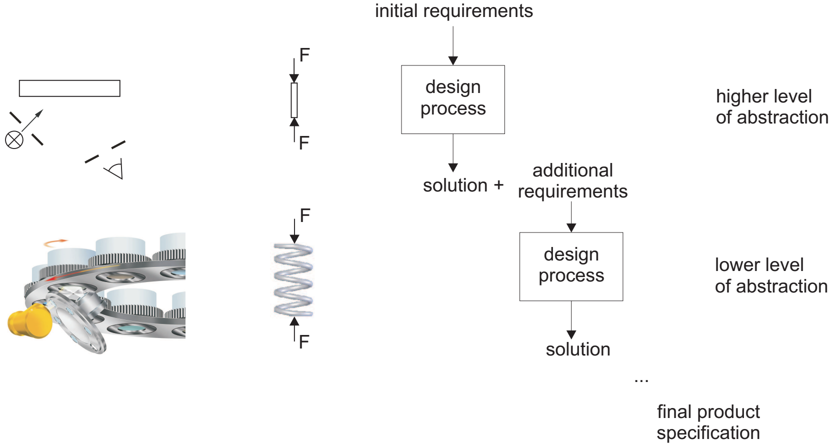


Figure 3.3: *levels of abstraction (after [26])*

After one sub-set has been addressed, the scope is widened to include the next (lower) level of abstraction. The solution from a previous level forms part of the input, together with a new set of requirements. The right-hand side of Figure 3.3 depicts this step-wise refinement from initial requirements to final product specification.

3.2 What is synthesis knowledge

In a design process, and more specifically the synthesis phase, one could say that the starting point consists of two things: a set of requirements and a parametric description of the embodiment that is not yet completely quantified. The end point is a description of a design that fulfills the requirements. It is *knowledge* that enables the designer to go from beginning to end.

The activity of synthesis is the translation of the under-defined input to fully-defined output. Knowledge rules provide methods to determine values for the unresolved parameters. These methods can vary from well-informed decisions to more random guesses.

After a value is determined for a parameter, a check is made to verify if the process is moving in the right direction. Again, knowledge provides the means to check if the current set of parameter values is allowed or conflicting. An allowed combination of values describes a valid (partial) design and the next parameter value can be resolved. If the values are conflicting, the embodiment description is invalid. One possibility is go back some steps and try a different route. The synthesis activity is finished if all parameter values are resolved and without conflict.

So far, three things are needed to describe synthesis:

1. the modeling entities: parameters;

2. methods to resolve a parameter value, i.e. reduce the dimension of the problem space: R-rules;

3. methods to limit the allowed solution space: C-rules.

This description of knowledge is labeled PaRC: Parameter, Resolve and Constrain. A relatively simple algorithm is used to simulate the activity of synthesis. The algorithm consists of three basic steps:

1. step forward: execute one R-rule;

2. check: execute the C-rules;

3. in case of a conflict, reverse one or several steps and try a different route.

The steps are similar to the Role-Limiting Method [45] and Backtracking from constraint satisfaction [22], but now explicitly defined in terms of the PaRC model. A more detailed formulation of PaRC is given in Section 3.3.

Note 1: resulting from these definitions, a difference exists between regular algebraic equations and R-rules: Expression (3.1) is not synthesis knowledge for parameters p_1 and p_2.

$$p_1 + p_2 = 1 \tag{3.1}$$

Equation (3.1) needs translation into either (3.2) or (3.3) before it is used to “step forward” in the synthesis process. Expressions (3.2) and (3.3) are synthesis knowledge, because they describe a method to resolve a parameter. Executing these R-rules results in a step toward the endpoint of the synthesis process.

$$p_1 = 1 - p_2 \tag{3.2}$$

$$p_2 = 1 - p_1 \tag{3.3}$$

Note 2: please note the difference between constraints and constrain (without "t"). In general, constraints are all restrictions and relations a solutions has to satisfy: both equalities and inequalities. In this thesis I separate equalities and inequalities. I do this by introducing R-rules and C-rules. The R-rules are used to model the equalities. C-rules are the relations that limit, restrict or *constrain* the allowed values of a parameter: inequalities.

3.3 A model of synthesis knowledge

Synthesis knowledge consists of a parametric embodiment model and a set of knowledge rules that enable generation of a valid embodiment. Both topics are discussed in more detail in the following section, and the algorithm that automates synthesis is discussed afterward [39]. I am aware of the fact that, at some points, I slightly abuse the formal mathematical notations, but I hope this does not cause confusion.

First, a general remark about the complexity of design automation is made, to illustrate the kind of calculations involved.

3.3.1 Challenges

Generating designs is generally an under constrained problem: there are more unknown parameters than there are equations. Actually, having equations is already a luxury. Studying design cases from engineering handbooks and industrial settings shows a mix of linear and non-linear algebraic equations, logic reasoning and random estimations. These relations can come from scientific formulas or fuzzy experience knowledge. Choices made earlier can influence what rules to apply later on. Designs with topological freedom can include or exclude entire groups of variables and relations. From a mathematical point of view, automating such a problem in a generic manner is not straight forward.

The parameters that describe a design are a mix of continuous and discontinuous, booleans and set-based parameters. For example, the parameter "material" is composed of a Young's module, density and melting temperature. One cannot use an *equation* to determine a value for the Young's module: such a material is not guaranteed to exist.

Another challenge with automating synthesis is the unpredictable input information. The user might, or might not, prescribes the value of certain parameters. And this can change each time the software is used. The algorithm could encounter systems of equations, or worse: a system of relations with all the mixed types described earlier. And in advance one does not know if a solution exists or not. In any case, the algorithm that performs synthesis has to complete the rest of the design automatically and find solutions, if they exist.

In summary, automating synthesis is a complex problem. Fortunately, there is a bright side: humans can do it. A human designer is able to generate solutions for the problem described above. This might sound trivial, but has an important consequence for the definition of synthesis knowledge and the algorithm proposed in Section 3.4.

3.3.2 Embodiment

An embodiment is modeled as a collection of parameters $p \in P$. Parameters that belong together are grouped within elements $e \in E$, so that the element "compression spring" is described by parameters such as material, wire thickness and length. Each parameter within an element is unique.

Each element $e \in E$ is an instantiation of an element *type* $t \in T$. All elements that are used during synthesis are first defined as an element type. Separate types exist for cog wheels, levers, springs and so on. During synthesis, multiple element of the same element type are allowed, e.g. two compression springs in one machine. Both elements are of the same type (i.e. compression spring) but have different instances. Although these elements have the same parameters, these parameters can have different values.

The parameter values of element e are represented by the following vector:

$$v_e(p) \quad p \in P_e \tag{3.4}$$

At creation of the element, the value is unknown. During synthesis, its value becomes resolved. The notion for the value being unknown or known is denoted as:

$$v_e(p) \begin{cases} = \lambda & \text{if } v_e(p) \text{ is not yet known} \\ \neq \lambda & \text{if } v_e(p) \text{ is known} \end{cases} \tag{3.5}$$

Once a parameter is resolved, its value can be valid or invalid. The C-rules determine the allowed solution space c_p for parameter p. The solution space c_p is formed by the C-rules that apply to parameter p and depend on $v_e(p) \neq \lambda$, i.e. all C-rules that can be executed to constrain the solution space.

Section 3.3.3 discusses how these parameters receive their values and how the solution space is determined.

Topology

A topology of the embodiment resembles the product structure. The topology is defined as a hierarchical tree of elements $e \in E$, with $E \subset T$. During generation of a topology, each element e is expanded with a (possibly empty) set of sub-elements $s_e \subset T$. The rules that prescribe the addition of sub-elements are termed X-rules. The X-rule changes the state of the sub-elements, denoted as:

$$s_e \begin{cases} = \lambda & \text{if } s_e \text{ is not yet known} \\ \neq \lambda & \text{if } s_e \text{ is known} \end{cases} \tag{3.6}$$

These sets of sub-elements have to respect the normal tree semantics, i.e. no element can be its own sub-element, neither directly, nor indirectly. Furthermore, each element has a unique super-element, except for a single root element that represents the complete design. A (partial) embodiment is thus represented by the set of elements E, their hierarchical structure s and the parameters values v, denoted as vector (E, v, s).

Solution

A solution for the synthesis phase is an embodiment that satisfies the following conditions:

$$s_e \neq \lambda \quad \forall e \in E \tag{3.7}$$

$$v_e(p) \neq \lambda \quad \forall e \in E, p \in P_e \tag{3.8}$$

$$v_e(p) \subset c_p \quad \forall e \in E, p \in P_e \tag{3.9}$$

I.e. the topology is fully expanded (3.7), no parameter value is unresolved for any element (3.8) and each parameter value lies within the allowed set, for each element (3.9).

3.3.3 Knowledge rules

The synthesis knowledge is organized using the object-oriented paradigm [4], where the objects are parameter $p \in P$ and element type $t \in T$. This means that parameters and element types are self sustaining entities, or agents, with their own knowledge rules. It is important to note that these rules are the same for every instantiated element e from t, i.e. two compression springs possess the same knowledge.

The general structure of a knowledge rules is as follows:

- object(s): the parameter(s) or element type(s) to operate upon;
- conditional set: the set of parameters that is required to have a value before the rule can be executed;
- action: the explicitly described operation on the object(s).

Three types of knowledge rules exist: X-, R- and C-rules. The rules are discussed in more detail in the following section, each with a pseudo-code example.

Note: as mentioned before, please note the difference between constrain*t*s (with "t") and constrain (without "t"). In general, constrain*t*s are all restrictions and relations a solutions has to satisfy: both equalities and inequalities. In this thesis I separate equalities and inequalities. I do this by introducing R-rules and C-rules. The R-rules are used to model the equalities. C-rules are the relations that limit, restrict or *constrain* the allowed values of a parameter: inequalities.

X-rules X_t An element type $t \in T$ contains the knowledge to add sub-elements. X-rules expand the hierarchy of elements by transforming a partial embodiment (E, v, s) into a new partial embodiment (E', v', s'). This is done by determining the set of sub-elements s'_e for an element $e \in E$ for which $s_e = \lambda$ and adding these elements to E to form E'. A single X-rule can add multiple types of elements, or multiple instances of the same element type.

Example An example of an X-rule is the topological expansion of element E_1 with N element instances E_2 from element type ET_2. In addition, the parameter *thickness* of the sub-elements E_2 must have the same value as the parameter *width* from E_1.

The pseudo-code is executed by the object element E_1. The "belief" parameter (value between 0.0 and 1.0) describes whether or not a rule is executable and with which certainty.

```
// CONDITION: E1 not expanded, N is resolved, width is resolved,
if ((E1.IsNotExpanded                  ) &&
    (E1.GetParameter(N).IsResolved     ) &&
    (E1.GetParameter(width).IsResolved ))
{
   // X-rule can be executed
   belief = 1.0
}
```

The action part of the X-rule is:

```
// ACTION: create N sub-elements
for(i = 1 to N)
{
   // instantiate element E2
   element E2 = new ET2.Instantiate()

   // set specific value for thickness
   E2.GetParameter(thickness).Value =
       E1.GetParameter(width).Value

   // add to E1
   E1.Add(E2)
}
```

R-rules R_t A parameter $p \in P$ that belongs to element type t has one or more knowledge rules to resolve its value. The parameter value of parameter p for some element $e \in E$ is fixed from its state λ in v. The result is a partial embodiment (E, v, s) that is transformed into a new partial embodiment (E, v', s)

that has less unresolved parameters. This implies that R-rules can only be applied to parameters that have not been fixed before.

Example Consider the R-rule from Equation (3.10) to resolve parameter p_1, which belongs to element E_1.

$$p_1 = 2 \cdot p_2 \tag{3.10}$$

The condition parts is executed by parameter p_1 to check if the rule can be executed, or not.

```
// CONDITION
if ((E1.GetParameter(p1).IsNotResolved  ) &&
    (E1.GetParameter(p2).IsResolved     ))
{
   // rule can be executed
   belief = 1.0
}
```

If parameter p_1 is selected for resolving, the action part of the R-rule is executed.

```
// ACTION
E1.GetParameter(p1).Value =
   2 * E1.GetParameter(p2).Value
```

Random generator The random generator is a universal R-rule that is applied if no other R-rule can be executed. The random generator can be used for many different types of parameters. In case of floating point or integer parameters, it generates a value within the allowed solution space. In case of set-based parameters such as material, it randomly selects one of the allowed materials.

Only embodiment parameters are allowed to be randomly generated. Embodiment parameters represent the degrees of freedom of a design, and exploring the design possibilities is done by varying the values of embodiment parameters. Random generation of performance is not allowed because these must be calculated by the analysis method. Random generation of scenario parameters is also not allowed, because it would be meaningless to compare the resulting performances (i.e. an embodiment has better performance because the scenario is lighter).

Random generation is only used if no other R-rule can be executed. If a parameter can be calculated by an equation, this has precedence over the random generator. The R-rule for random generation looks similar to a R-rule, with the difference that no other parameter values are required to be resolved. Consider the random generation of a value for parameter p_1 within the previous element E_1. The condition part is:

```
// CONDITION
if ((E1.GetParameter(p1).IsNotResolved)
{
   // random generation rule can be executed
   belief = 0.1
}
```

Note the value for belief is 0.1. This signals that it is possible to execute the rule, but *not* with the same certainty as the R-rule from Equation (3.10). If no parameter or element has a rule with belief 1.0, the action part of the random generation rule could be executed:

```
// ACTION
// get lowerbound and upperbound of solution space
min = E1.GetParameter(p1).Lowerbound
max = E1.GetParameter(p1).Upperbound

// generate random value between min and max
value = random(min, max)

// resolve parameter
E1.GetParameter(p1).Value = value
```

C-rules C_t A parameter $p \in P$ that belongs to element type t has knowledge rules to check if its own value lies within the constrained, allowed solution space. C-rules govern the boundaries of what is feasible within a design problem. It returns a set of allowed values for a parameter: c_p. The parameter checks if the set has members and, if the parameter has a value, whether or not the values lies within that allowed set. At any time during the synthesis process, the value of any assigned parameter must be a member of all sets of allowed values produced by applicable C-rules.

Example C-rules constrain the allowed solution spaces for parameters. In case of a parameter with double value (e.g. length, width), the solution space is defined by a lower bound and upper bound. Because multiple C-rules can constrain one parameter, only the most constraining upper and lower bound is relevant. Consider Inequality (3.11) for parameter p_1.

$$p_1 \geq p_2 \tag{3.11}$$

The C-rule can be executed independently of the value for parameter p_1. The condition part for the C-rule is, derived from Inequality (3.11):

```
// CONDITION
if ((E1.GetParameter(p2).IsResolved)
{
   // rule can be executed
   belief = 1.0
}
```

And the action part is:

```
// ACTION
// new value for lowerbound
newLowerBound = E1.GetParameter(p2).Value

// is most constraining?
if(newLowerBound > E1.GetParameter(p1).Lowerbound)
{
   // TRUE: newLowerBound is most constraining
   E1.GetParameter(p1).Lowerbound = newLowerBound
} else
{
   // FALSE: newLowerBound is not constraining
   // original Lowerbound remains
}
```

Note 1: the “action” part is not necessarily a deterministic mathematical algorithm; it could also take the form of a fuzzy logic system or external application. A complete reasoning system can be located in this “action” section.

Note 2: the “belief” concept is just one way to discern between resolving through equations or with random generation. More tuned use of the belief concept is also possible, e.g. to take into account in how many R-rules a parameter occurs.

3.4 Synthesis algorithm

A view from cognitive design research is that “design is most appropriately characterized as a construction of representations. The initial representation is formed by the requirements, and through a series of transformations (e.g. replicate, add, detail, refine, modify and substitute) develops toward its final form” [52]. The order in which parts of the representations are modified is described by a strategy. One human design strategy is called the“structured decomposition strategy”, where the predictable paths from input to output are used during synthesis.

Another strategy is described as opportunistic, where it depends on the current state of the design and available knowledge to decide on that moment what to do. The particular non-systematic character is attributed to the fact that designers, rather than systematically implementing a structured decomposition strategy, take into consideration the data that they have at the time. This focuses on their knowledge, the state of their design in progress, their representation of this design and the information at their disposal [52].

This section describes a synthesis algorithm that executes R- and X-rules using an opportunistic strategy, based on a design mechanism used by a human designer. Each step considers the available knowledge rules and current parameter information to decide what to do. The algorithmic description of this process is divided into a number of steps, beginning with initialization. The algorithm is also described in [3] as a constraint satisfaction solver for constraint programming.

Step 0: initialization The algorithm initialization requires a set of element types T. A single instance is denoted the root element, from which the algorithm starts. The parameter requirements of an element e are super-imposed on the knowledge rules when it is instantiated.

These user imposed requirements are (combinations of) rules without a conditional set, i.e. always and immediately executable. Examples are: length $= 10.0$, thickness ≤ 5.0, M4 $\leq$ bolt size $\leq$ M20, material $\neq$ copper, aluminum, gold.

Loop step 1: constrain check: test if any parameter has a conflict. Execute C-rules C_t if the conditional set allows it. After execution of the C-rules, test for all parameters:

1. solution space not empty: $c_p \neq \emptyset$;

2. if $v_e(p) \neq \lambda$: value lies within allowed set: $v_e(p) \subset c_p$.

If all tests are passed positively: current embodiment representation is allowed. If one test is negative: current embodiment is not allowed. When possible, a previous (allowed) representation (E, v, s) is retrieved to proceed.

Loop step 2: completeness check: check for complete embodiment. Test for all elements $e \in E$:

1. $s_e \neq \lambda \quad \forall e \in E$;

2. $v_e(p) \neq \lambda \quad \forall e \in E, p \in P_t$

This checks if (1) the topology is fully expanded, and (2) no parameter value is unresolved for any element. If these tests are passed, the embodiment (E, v, s) is a complete representation of the embodiment and the synthesis phase is terminated.

Loop step 3: advance partial embodiment: execute one R- or X-rule. This is done in three phases:

1. explore possibilities;
2. select parameter or element;
3. execute rule.

The first step tests the conditional sets for the R- and X-rules of each parameter and element, and gathers the possibilities. Second, one parameter or element is selected to be resolved or expanded. Using the concept of belief, a difference is made between rules with and without random generation: rules without random generation have a higher belief (equations before guesses). Selecting which parameter or element to be resolved can be made with or without a strategy. In the most basic form, this is a random process.

As the third step, the action of the selected rule is executed, after which the algorithm continues with loop step 1: constrain check.

Note 1: if backtracking to previous partial embodiments is implemented to solve C-rule violations, the algorithm effectively implements a non-deterministic version of depth-first search.

Note 2: no effort is taken to trace the origin of a conflict, because this can be several R-rules back inside a different element. Given the diversity of parameter and knowledge rules, as well as the "random walk" of the algorithm, this topic is reserved for future work.

3.5 Limitations

The PaRC model and the algorithm from Section 3.4 are used to automate synthesis of parametric engineering design, with the following limitations.

3.5.1 Systems of equations

The algorithm does not solve systems of equations. Developing a generic (autonomous) solver for a mix of linear, non-linear, logic relations for continuous and discontinuous parameters is outside scope of this thesis. However, the presence of a system of equations is recognized by observing if a parameter can be resolved by more than one R-rule. This signals that a system is encountered, and the current embodiment is possibly incorrect and therefore abandoned. An example of this approach is given.

Consider Equations (3.12) and (3.13) with two free parameters p_1 and p_2.

$$p_1 + p_2 = 2 \tag{3.12}$$

$$p_1 \cdot p_2 = 1 \tag{3.13}$$

These equations must be solved as a system in order to yield the only valid solution $p_1 = 1$ and $p_2 = 1$. However, the nature of the algorithm is that it resolves one parameter at a time. At the initial state, neither parameter can be resolved. The algorithm will randomly generate a value for one of the two parameters: e.g. $p_1 = 0.5$. Next, the algorithm will explore the possibilities, and finds it can resolve p_2 by R-rules derived from either (3.12) or (3.13). This leads to $p_2 = 1.5$ or $p_2 = 2$, both incorrect.

The erroneous situation is prevented by observing the fact that p_2 can be resolved by *more than one* R-rule (excluding random generation). If such a situation is encountered, the entire embodiment is abandoned (the “better safe than sorry” paradigm is applied). A remedy for this situation is to add an additional R-rule that recognizes specific systems of equations and solves them explicitly.

If one requires the algorithm to prevent systems of equations, one must state *all* parameters of an equation as possible resolve options. Even if no computational method is included to actually resolve the parameter. This is done because they *could* be part of a system, and this situation must be noticed.

3.5.2 Consistency and solvability

The consistency and/or solvability of a PaRC model is not explicitly checked. Therefore, it is possible to model ambiguous or erroneous situations and run the algorithm. The algorithm will try to generate embodiments with the given model.

Autonomous analysis of the PaRC model would require the content of the rules to meet some prescribed (formal) expressiveness. An algorithm would be executed to detect an over-constrained situation or the lack of any solution. Both checks would be valuable to execute during the knowledge engineering phase, but are outside scope of this thesis.

The algorithm from Section 3.4 only detects systems of equations (an over-constrained problem), but does not solve them. The existence of solutions is checked by running the algorithm and waiting for a solution.

3.5.3 Revising decisions

The model does not allow revision of earlier made decisions. Parameter values that have been resolved cannot be changed. Neither can sub-elements be expanded twice. The reason is that a parameter value could influence decisions made later on during synthesis. To prevent the risk of a loop, the action of resolving or expansion twice is not allowed. Instead, an "estimation" parameter can be introduced to estimate a first value, after which another parameter describes the final value.

The consequence of "forward only" generation is also that the topology is static: after sub-elements are added to an element, no modifications are made. The random-walk during synthesis creates differences in the topology, but informed adjustments are not made.

Adjusting an embodiment to better meet performance criteria is outside the scope of this thesis. Recommendations concerning this topic are made in Section 7.2.

3.5.4 Algorithm

The disadvantages of the backtracking algorithm are known [3]. The risk of repeated failure due to the same reason and the lack of memory for conflicting values of parameters are among the main risks of inefficiency. The fact that conflicts are not detected before they really occurs is another disadvantage. However, its robustness and the fact it is independent (to certain extent) of the content of the rules make it an attractive baseline algorithm.

Chapter 4

Knowledge engineering method

This chapter describes a method to decompose and model a design process and the knowledge in terms of PaRC. The method begins with a design process and one or more knowledge sources. Modeling a design process requires identification and typing of information entities, sub-processes and knowledge rules with nomenclature from the original source.

The method aims to aid the "what to look for and how to get it"-aspect of knowledge acquisition. In case of a human source, the knowledge engineer acts as facilitator to make the expert's knowledge explicit.

The first step divides a design process into levels of abstraction. After a suitable level is selected for further modeling, the analysis method is formalized and used to identify the embodiment and scenario. Subsequent formalization of the synthesis knowledge leads to a completed PaRC model.

The outline of the method is depicted in Figure 4.1. The individual steps are elaborated in the following sections.

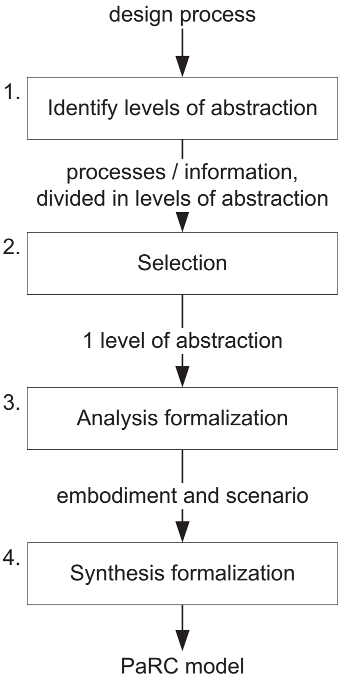


Figure 4.1: *knowledge engineering*

4.1 Step 1: identify levels of abstraction

The first step to find the levels of abstraction is the identification of the performance parameters. Levels of abstraction emerge as performance parameters of equal importance.

The performance parameters are found by asking an expert to describe the quality of a design, or what parameters decide the quality? Give a design to an expert and ask how “good” it is: what aspects does he/she consider? Give two different designs to an expert, and ask which design is the better one: again, what aspects are considered? What information decides if a design is *not* approved?

These questions are asked at several moments during the design process. The goal is to obtain a list of performance identifiers. Examples are cost, weight, maximum stress, optimal speed, first eigenfrequency, dynamic response and stability.

After the performance parameters are known, identify levels of abstraction by grouping the performance indicators of equal importance and their analysis methods. Performances of higher importance are identified early on in the design process: on higher levels of abstraction. Performances of equal importance belong in the same level. When multiple levels exists, rank these relatively to each other. To complete the division of a design process into distinct levels of abstraction, state what the input and output information is for each level.

4.2 Step 2: selection

It is possible that the design process is traced from first conceptual design all the way to detailed design for manufacturing. Many levels of abstraction are identified, some of which are not suitable for modeling with PaRC. This step selects the parts that are suitable, using the following check:

1. the analysis methods are known;
2. the degrees of freedom are known and a knowledge source is available (i.e. the problem is well-structured);
3. the design process can be described with parametric information and quantifiable data.

It is advisable to select entire levels of abstraction to maintain coherence of the decision making process. If a group of performance parameters is taken into account to make a decision, these parameters should be of equal importance and provide a complete image of the quality of a design.

4.3 Step 3: analysis formalization

A design process is selected for further formalization. The current step involves the formalization of the analysis methods that quantify the performance parameters. This is done by asking "how" these performance parameters are calculated: the analysis method.

If an explicit method is used, describe the input and output information as parametrically as possible. After all parameters are identified, begin typing each as embodiment, scenario, auxiliary or performance. To decide which type a parameter is, the following decision scheme is suggested:

1. embodiment: parameters and/or topological structures that a designer is allowed to manipulate;

2. scenario: description of a usage situation, often prescribed by external parties or a previous level of abstraction. A designer is *not* allowed to freely manipulate these parameters;

3. auxiliary: can be substituted by either *only* scenario, or *only* embodiment. Auxiliary parameters are the dependent parameters, while embodiment and scenario are independent;

4. performance: calculated by a *combination* of embodiment and scenario.

This decision scheme is intended as a guideline. It should always be used in combination with the intended common sense understanding of embodiment, scenario and performance: an embodiment is the designed artifact or system, and scenarios are imposed from "outside". Performance comes from combining an embodiment and scenario, and states something about the quality of a design. Auxiliary parameters are used to rewrite other parameters, or give some additional information. These can be substituted without changing the "core" of the design process.

After all parameter have been typed, the embodiment descriptions from multiple analysis methods are combined to give a complete embodiment on a single layer of abstraction.

It is not obliged to "keep" all parameters, as some can be eliminated by substitution. All embodiment and scenario parameters are kept and only the major auxiliary parameters. Including more or less parameters creates a trade-off between the expressiveness of the model and the (computational) complexity.

Analysis methods occur in a number of different ways. Section 4.3.1 discusses several differences between simulation, formula and tacit analysis methods and how to cope with them during decomposition.

Note: during the identification process of these parameters, be careful not to focus too much on a single exercise: an embodiment requirement (e.g. length = 10) can be mistaken for a scenario parameter. Vice versa, a scenario can be mistaken for a degree of freedom: how much load can this bridge take? When modeling a design process by focusing on a single exercise, there is the risk of extrapolating the exercise to the entire process. During one exercise, an expert is likely to organize his/her activities in such a way that the expected difficulties are addressed properly. However, the next exercise might have other difficulties, making the expert change him/her strategy. And the next exercise could change again.

4.3.1 Differences in analysis methods

As discussed previously, an analysis method calculates a performance parameter based on an embodiment and scenario. Three common types of analysis methods are discussed in this section.

Simulation

Simulations are used to reveal performance issues that are hard, or impossible, to quantify analytically. Finite element analysis is a popular example of such simulations, such as the crank shaft depicted in Figure 1.2a.

Before a simulation is executed, the embodiment model has to be defined. This can be an abstracted representation, such as two-dimensional wire frame model to represent a bridge, or a detailed three-dimensional model of the same bridge. The scenario can be a static or dynamic load case: a frequency sweep to reveal eigenfrequencies, or simply gravity to see if a structure fails under its own weight.

The crank shaft of Figure 1.2a shows a simulation of the material stresses under mechanical and thermal load. Whatever embodiment the designers come up with, these are analyzed under the same load and temperature scenario.

The model of the crank shaft is in this example the embodiment and the load and temperature the scenario. All of these parameters have to be specified before a simulation is ran to reveal the material stresses. An analysis simulation cannot be inverted to determine an embodiment parameter.

Formula

Formulas can occur in parametric design problems during all sub-processes. After the performance parameters have been identified, their means of quantification are the analysis methods, which have the general layout of Equation (4.1).

$$< performance >= f(embodiment, scenario) \tag{4.1}$$

For instance, the design of a spring element. The design goal is to exhibit a specific force upon a fixed displacement, using Equation (4.2) to calculate force

F if a displacement s is imposed on an artifact with stiffness k. The parameter types are: F is performance, k is embodiment and s is the scenario.

$$F = k \cdot s \tag{4.2}$$

However, Equation (4.2) can also be used to calculate the parameters k and s. A characteristic of algebraic equations is that they can be re-written and used whichever way is most convenient. But the use of an equation does not change the fact that it is an analysis formula to calculate performance.

Tacit

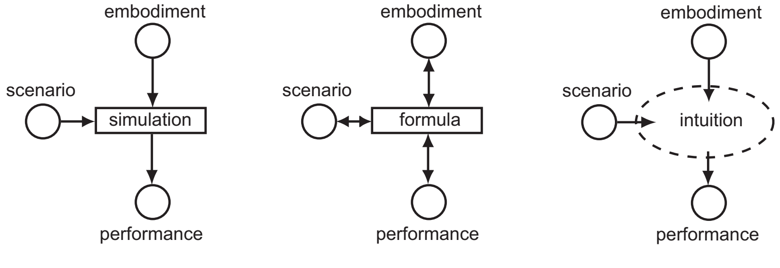
Experts often do not use calculations or simulations to judge the quality of a design. Instead, it is done with an "expert's eye" to intuitively decide to proceed, abort or adjust a design.

Judging the aesthetics of a house requires the object to be described in terms of colors, shapes and positions. These specifications are not required for each screw and door-knob, but only the roof, windows and walls. The people to purchase the house require a certain style, and the aesthetics are judged in this context.

Although this analysis method is hard to make explicit, it can still be used to describe and model the design process, albeit perhaps not as parametrically as a formula.

Differences between simulation, formulas and tacit analysis

The differences between simulation, formula and tacit analysis method are illustrated in Figure 4.2. To use these methods as decomposition instruments requires their input and output sets of information to be explicit. Identifying these sets is done differently for the three types of analysis methods.



(a) simulation (b) formula (c) tacit

Figure 4.2: *three types of analysis*

Simulation-analysis has a clear relation between the input and output: the input has to be defined, after which a simulation is ran to reveal the performance.

This is used directly to distinct between the performance information and scenario and embodiment information.

Formulas are explicit input-output relations, with the added difficulty that formulas can be re-written algebraically, so what is the input and what is the output? For example, Equation (4.3) can be written as (4.4), (4.5) and (4.6). From observation of the equations, it is not clear which one is the analysis expression that is needed for analysis-oriented decomposition.

The analysis expression is the one which is an explicit description of a performance parameter (see Expression (4.1)). The left-hand side states the performance and the right-hand side contains the scenario and embodiment. It depends on the design context which performance is relevant and which expression is the appropriate analysis formula.

$$p_1 + p_2 + p_3 = 0 \tag{4.3}$$

$$p_1 = -p_2 - p_3 \tag{4.4}$$

$$p_2 = -p_1 - p_3 \tag{4.5}$$

$$p_3 = -p_1 - p_2 \tag{4.6}$$

Tacit analysis methods lack explicit input-output relations. However, the information can be divided into the three types. Once the performance information is known, one asks “how” this is determined. The answer from the expert contains the scenario and embodiment information. Similar to formulas, the goal is to obtain a description of “performance is determined by . . . ”, where the right-hand side contains scenario and embodiment information.

4.4 Step 4: synthesis formalization

This phase uses the previously found parameters of the embodiment to acquire and model the knowledge rules for synthesis. The embodiment parameters are the starting points to state the R-, C-rules and finally the elements and X-rules.

If the knowledge source mentions other parameters during knowledge acquisition, these can be added to the group and included in the acquisition process. Each parameter is made explicit with a name, short description and type (embodiment, scenario, performance or auxiliary), such as stated in Table 4.1. The table contains part of the embodiment of an industrial design case, discussed in more detail in Section 6.2.1

Table 4.1: *optical chamber, parameters (partial)*

type	name	description
scenario	S_{mat}	sample material
	T_A	tube surface area
	T_{mat}	tube material
	Det_A	detector surface area
embodiment	T_x	tube position, x
	T_y	tube position, y
	D_{mat}	diaphragm material
	...	...
auxiliary	$OC_{\beta P}$	angle primary axis
	OC_{LP}	length primary axis
	E_{max}	maximum energy
	...	...

The knowledge acquisition activity is one where questions are asked and answers formulated. The questions should lead to useful answers. The following section provides more detail how to acquire the relevant knowledge.

R-rule To obtain the R-rules for an object, specific answers are searched for: the *condition* statement and the *action* procedure (discussed in Section 3.3.3).

The general form of a R-rule for a parameter is:

```
if( <condition> )
{
    <parameter value> = <action>
}
```

The condition and action statements are acquired by formulating an explicit answer to the following questions:

1. condition: *when* can you calculate a value for this parameter?

2. action: *how* do you calculate the parameter value?

Example Expression (4.7) shows the action part of the R-rule to calculate the x-coordinate of the tube (embodiment parameter T_x), Figure 4.3.

The action part of the R-rule to choose a diaphragm material is given in Expression (4.8). This rule selects material A if the value for E_{max} exceeds value V, else material B is selected.

A parameter value can also be randomly selected between an upper and lower bound. For instance, the parameter $OC_{\beta P}$ is randomly resolved between A and B, Expression (4.9). The user has certain default values for A and B, or these values follow from other calculations.

$$
\begin{aligned}
T_x = & \quad OC_{(x)} - OC_{LP} \cdot \cos(OC_{\beta P}) && (4.7)\\
D_{mat} = & \quad if(E_{max} > V) && (4.8)\\
& \quad \{D_{mat} = A\} \\
& \quad else\{D_{mat} = B\} \\
OC_{\beta P} = & \quad random(A, B) && (4.9)
\end{aligned}
$$

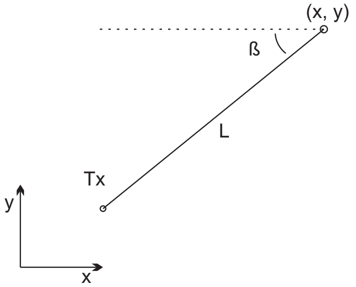


Figure 4.3: *tube position*

If there are exceptions or fuzzy decisions, these can be included in the rule. Multiple R-rules are stated if a parameter can be resolved in more than one way.

The condition part tells *when* a rule can be executed: what information must be known? This follows from the content of the action part: all information that is mentioned must be known before it is used.

C-rule The C-rules are found by inquiring, for each parameter, if the value is always good and never needs validation. If it is checked somewhere during the synthesis phase, this signals the existence of one or more C-rule. Again, ask *how* and *when* this is done. The general form of a C-rule for a parameter is:

```
if( <condition> )
{
   <parameter value> is valid if: <action>
}
```

The condition and action statements are acquired by formulating an explicit answer to the following questions:

1. condition: *when* can you check the validity?

2. action: *how* do you check validity?

Example An example of a C-rule is the check if a value exceeds a certain minimum ratio: Expression (4.10) must be valid. This yields the action parts of C-rules (4.10),(4.11) and (4.12), for parameters a, b and c respectively. Either one of these three C-rules is sufficient to detect invalid embodiments, but for completeness all three are stated.

$$a \geq \frac{b}{c} \tag{4.10}$$

$$b \leq a \cdot c \tag{4.11}$$

$$c \geq \frac{b}{a} \tag{4.12}$$

An example of a non-algebraic C-rule for parameter “material” is when certain materials are excluded for specific environments. The condition of the C-rule is true if the environment is known. The action part will exclude all materials from a database that are sensitive to corrosion. The materials have a boolean to signal if they belong to the allowed set and this boolean can be made “false” by the C-rule, for example as shown in the following pseudo-code:

```
\\ CONDITION
if(E1.GetParameter(environment).IsResolved)
{
   \\ rule can be executed
   belief = 1.0
}

\\ ACTION
if(E1.GetParameter(environment).Value == corrosive)
{
   \\ exclude sensitive materials
   foreach(material in database)
   {
      if(material.CorrosionSensitive)
      {
          material.IsAllowed == false
      }
   }
}
```

Element types Element types are groups of parameters that are addressed simultaneously. The question at this point is: what parameters belong together?

The product that is being design can be used to determine the element types as physical entities, such as springs, belt drives, x-ray tube and detector.

Element types can also be defined according to different functions they perform in a design. For example, an optical chamber (Section 6.2.1) has two diaphragm sets: one between tube and sample, and the other between sample and detector. Both sides have different functions and are modeled as different element types. The tube side must radiate the sample surface homogeneously, while the detector side must focus on the center part of the sample. Both sides position their diaphragms differently. Instances of the element type "diaphragm" will occur as sub-elements on both the tube and detector side.

The Vanderlande Industries case (Section 6.2.2) has element types specifically to model product functions such as check-in, screening and sorting. Each of these main functions is a different element type, and within the main function are element types for specific sub-functions, Figure 4.4. For example, the function to gather incoming baggage flows, to redistribute the flow and make it evenly distributed, to feed baggage in and out of the processing equipment and to present it to the following sub-system. The element "belt" is used inside each functional element, but connected differently each time. Different element types are made for different functions, and each function places its belts with specific knowledge.

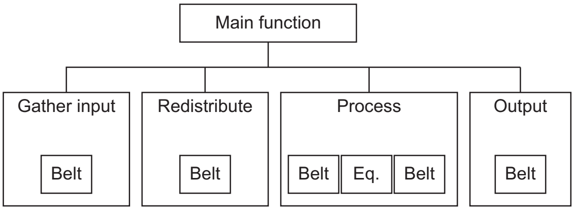


Figure 4.4: *element types for functions*

X-rule X-rules are revealed as the process that is executed to expand the topology of the embodiment with new elements. Once the element types are identified, the X-rules are formulated to add them.

An X-rule is executed when sufficient information is known to connect the sub-element(s) to existing parameters. After the expansion, each element will be self-supporting but must "know" its place within the topology. The layout of an X-rule is:

```
if ( <condition> )
{
   \\ 1. create sub-element instances
   \\ 2. connect to existing embodiment
   \\ 3. add to element
}
```

The condition statement of the X-rule must check for the required information to connect the sub-elements to the embodiment. Formulation of the X-rules depends on the chosen element types.

Note 1: knowledge rules can refer back to performance or scenario parameters. For instance, a performance requirement is stated at the beginning and influences decisions during synthesis. If such parameters are mentioned during synthesis, these are included in the knowledge acquisition process.

Note 2: the parameters are revisited several times, to make sure no R- or C-rules are left out. Existing knowledge acquisition techniques are used to make this process as efficient as possible.

Parameter dependency graph Synthesis knowledge is explicitly written as PaRC entities, but can also be graphically presented to visualize the parameter dependencies. Directed graphs are used to represent the network of parameters and R-rules. Parameter dependency graphs are used to observe more intuitively "what knowledge looks like".

A parameter dependency graph consists of nodes and directed edges. Nodes represent parameters and the directed edges describe R-rules. The direction of the edges are pointed toward the *target* object: the parameter to resolve. If node D has N edges directed toward it, this represents a R-rule for node D that requires the N other values.

An example of such a parameter dependency graph is shown in Figure 4.5. This graph represents the R-rules derived from Equation 4.13, which comes from a compression spring. The equation describes the winding ratio w as a fraction of the spring diameter D and the wire diameter d_{wire}.

$$w = \frac{D}{d_{wire}} \tag{4.13}$$

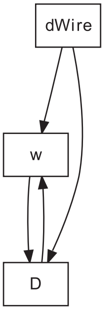


Figure 4.5: *parameter dependency graph, example*

Node w has two edges directed toward it, meaning that its value can be resolved if the values of D and d_{wire} are known. As shown in Figure 4.5, the parameter value of d_{wire} has no edges toward it. This means d_{wire} cannot be resolved with this equation. The reason is that the allowed values of d_{wire} are discontinuous (specified by a DIN standard), and a calculated value from an equation is not guaranteed to be exactly a DIN value. Instead, d_{wire} is resolved by another rule, or random generator, before Equation 4.13 can be used to resolve either w or D.

Note 1: The graphs are made using Graphviz win 2.16. The layout algorithm automatically aims edges in the same direction (top to bottom, or left to right) and then attempts to avoid edge crossings and reduce edge length (the so-called dot setting). Further analysis of these graphs is outside the scope of this thesis: it is only intended as a graphical representation of knowledge.

Note 2: The random generation rule is not taken into account. Parameters that do not occur in any R-rule are also not depicted.

4.5 The knowledge document

The result of the knowledge engineering phase is a model of the design process and synthesis knowledge that is described explicitly in a knowledge document. The model of the design process states the information flow through the different sub-processes and the synthesis knowledge describes the knowledge is relevant for synthesis. The latter is specified in PaRC entities of parameters, element types, X-, R- and C-rules.

The models within the knowledge document are not dependent of any single design exercise. The analysis method is used as guideline to determine the expressiveness of the models, not the expert nor knowledge engineer. The goal is to retrieve a static, reliable model of the design process.

The document contains the knowledge as far as relevant to execute the design process and synthesis. No explanations, causal relationships or new knowledge is included. However, these aspects can be added by using the PaRC model as starting point and gathering more data.

The knowledge document contains valuable expert knowledge that can be stored or used for training purposes. However, the models can also be used to develop design automation software. The following chapter will discuss the process of developing the software application.

4.6 Limitations

The knowledge engineering method is developed for engineering design with existing knowledge, parametric information and quantitative data. Modeling the parameters and their R- and C-rules is relatively strictly prescribed. The model can be modified to the taste of the knowledge engineer, but a first model is made quite quickly. At least, that is the experience of the author.

The scope further includes designs with topological degrees of freedom, such as assemblies and product systems. However, modeling topological entities requires more “personal creativity” of the knowledge engineer. These decisions are less strictly prescribed and therefor lead to more ad hoc modeling solutions. Although it is certainly possible to model designs such as transport networks, the goal of the method is to eliminate the ad hoc approach. The method is most suitable for parametric engineering with fewer topological degrees of freedom.

Chapter 5

Software development method

The development process of design automation software is prescribed for design cases that are unfamiliar to the development team. Guidelines are given about procedure, as well as several approaches for efficient software development. The previously described knowledge engineering method forms a substantial part of the process and is positioned relative to the other activities.

The software development process consists of steps depicted in Figure 5.1: the first step creates an overview of the major processes and products within the new company. Next, a section is selected for further study and model the design process using methods from Chapter 4. The PaRC model is subsequently automated and the user interaction is determined.

The steps are described in more detail in the following sections. Diverging from this plan is of course allowed, but being aware of a procedure or information structure is a great benefit during software development, as concluded from the development of several prototypes [40].

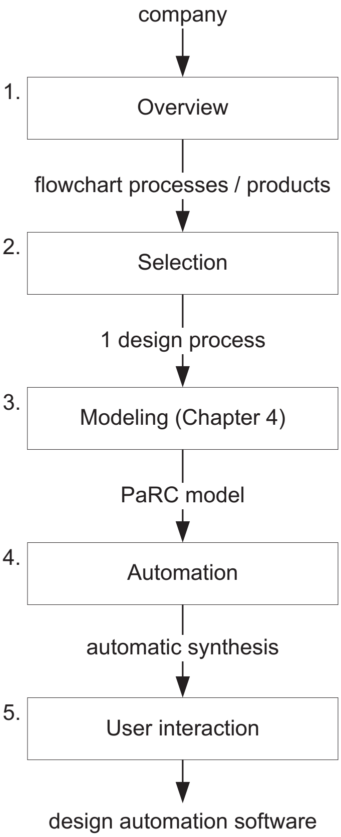


Figure 5.1: *software development*

5.1 Step 1: overview

The first step is to create an overview of the company and check for suitable design processes, getting to know the unfamiliar design environment, individuals and nomenclature. This phase has no strict procedure, as it depends greatly on the specific situation. The general goal is to create an overview of context surrounding the design of the product(s). Especially the input/output interfaces between departments, processes and activities are of interest.

A suggestion is to begin with a product family tree and process flowcharts from initial requirements to final product specifications. State the activities and sets of information as they evolve from beginning to end. Where possible, specify the design processes with respect to e.g. disciplines (physics, mechanical), components or level of detail. Different departments and/or expert designers might embody an isolated (part of a) process, as are chapters and sections in literature sources.

The output of this phase is an overview of the different (design) processes, relative to each other. The information is specified in the company's nomenclature. An overview can encompass multiple design processes and levels of abstraction from conceptual systems design to manufacturing planning.

5.2 Step 2: selection

Suitable design processes for software development are identified, using a suitability check that is similar to Section 4.2. In addition, the check here also takes into account the ability to be automated. Design processes are required to meet the following conditions:

1. the analysis methods automatically quantify a set of performance parameters;
2. (identical to Section 4.2) the degrees of freedom are known and a knowledge source is available (i.e. the problem is well-structured);
3. (identical to Section 4.2) the design process can be described with parametric information and quantifiable data.

5.3 Step 3: modeling

Developing a model of the selected design process and synthesis knowledge is done using the knowledge engineering method described in Chapter 4. First, levels of abstraction within the design process are identified, and the most suitable one selected. The information around the analysis methods describes the core expressiveness of the software.

Once the parameters around the analysis method are made explicit, a model of the design process is made. The content of the information sets of performance, scenario and embodiment are known. This also determines the functionality of the software modules for synthesis and analysis.

5.4 Step 4: automation and implementation

Automating the models obtained from the previous step involves choosing the right algorithm for the problem. If the synthesis knowledge is modeled in PaRC, the algorithm from Section 3.4 can be used. Automation of the remaining processes is very case dependent, but at least the input/output information is explicitly known.

The sub-processes and the information exchange between them is managed by a higher level framework: the architecture. The approach of this thesis is to build a *specific* software program for a *specific* design case. When building the next program, a certain amount of code is re-used, effectively reducing software development effort. The code being re-used in multiple software programs is the *generic* code, and the code that contains the design specific information and knowledge is the *specific* part. This notion led to the development of a generic software architecture, discussed briefly here but details are provided in [48].

A generic architecture defines the interfaces between generic and specific code, on a module level. Modules are larger building blocks of a system with a clear task. They provide a separation between interface and implementation components. The interface describes functions and data abstractions available to the world surrounding the module. It expresses the components that are provided and required by the module. The implementation encompasses the code that takes care of the tasks to be performed, internally. A module for visualization for instance, "talks" about geometric surfaces and colors; whereas a module for synthesis has interface components like parameters and R-rules. In modern programming languages like C#, modules can have multiple explicitly defined interfaces. A visualization module for instance can provide both a 2D and 3D interface.

If these generic sections are used from existing software, such as a CAD system, the specific parts act as plug-ins. The interface between generic and specific is in that case defined by the Application Programming Interface (API). The research project in which this thesis is executed developed its own generic architecture in the C#.NET environment. The architecture resembles the model of a design process and the interfaces between software modules are defined according to the information sets between different activities in the design process, discussed in Section 3.

The main paradigms of generic architecture development are summarized:

- modularity;
- centralized data model;
- separation of generic and specific code and data.

The generic architecture prescribes the standardized interfaces of modules. This enables other module versions with new internal implementations to be plugged-in. This enhances the tool building process where multiple developers work on the same system or where a customized version of a module is needed. The generic modules have generic implementations and generic interfaces. The specific modules have specific implementations and can have both generic and specific interfaces. This makes it possible to embed specific modules in a generic environment.

The generic part encompasses the libraries to support and manage data storage, gathering the user input, and presenting the output of the synthesis process back to the user. The specific code that is connected to the generic part includes modules that describe the different steps of a specific design case.

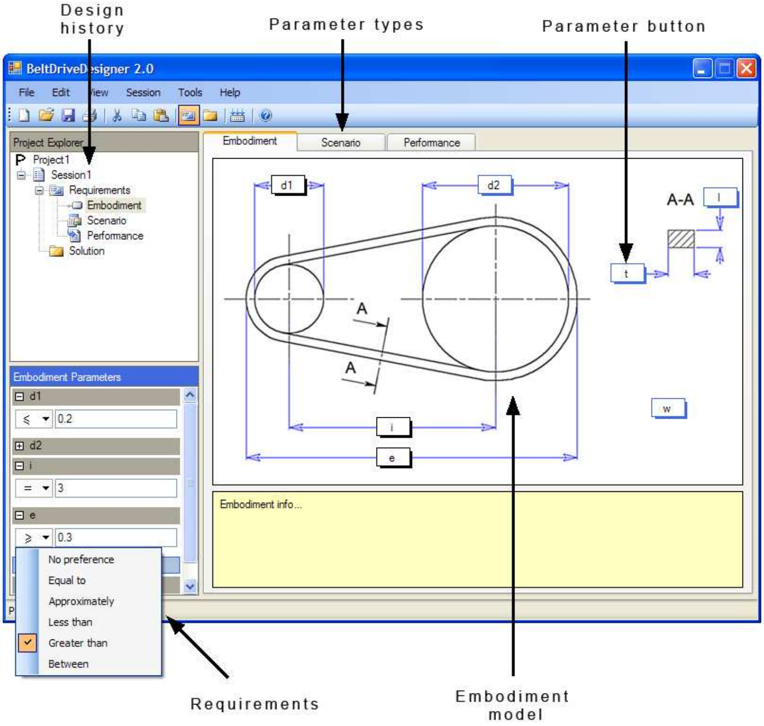
Chapter 6 discusses an implementation of the generic framework.

5.5 step 5: user interaction

The final step in software development is to decide the interaction between software and user. The graphical user interface (GUI) between user and software is the place to state the design requirements and observe what the program has done. Although the graphical appearance of this interface changes per case, the functionality is described using the prototype software for belt drive design.

Requirements input The requirements are entered using the interface depicted in Figure 5.2. The panel at the left-hand side of the GUI contains the Project Explorer that displays the design history, similar to the Windows Explorer. When the user tries different sets of requirements, new sessions are created within a project. Each session contains a requirement set, and a solutions set containing the results of multiple synthesis runs. The requirement node has three sub-nodes to provide direct access to the embodiment, scenario and performance requirements. Clicking the requirement node or the solution node will cause the program to switch to the corresponding graphics at the right-hand side of the GUI.

Below the Project Explorer is the “parameter input” window. This window contains fields to specify values for the parameters. The user can set the requirement type (e.g. equal to, less than, greater than, between) and provide a value or range. He/she can also leave the parameter free to be determined by the synthesis process, allowing a degree of freedom in the synthesis process. Located in GUI are some parameter buttons with the name of the parameter indicated in the sketch. The user can double-click these buttons and change the value of the parameter in a pop-up dialog, shown on the left bottom where requirements of d1, i and e are entered.



d1 ≤ 0.2

d2

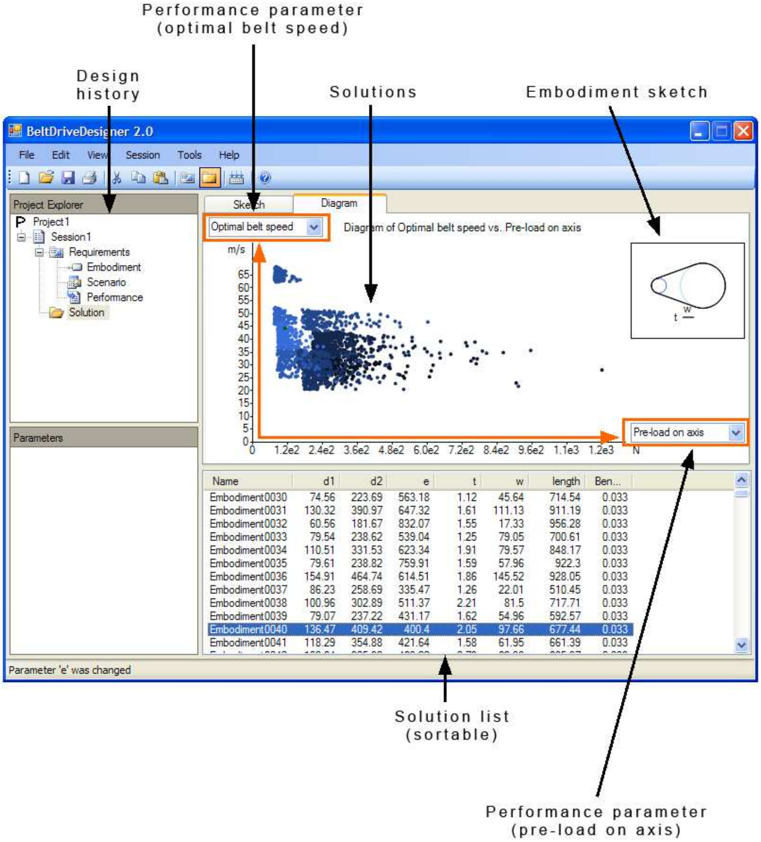
i = 3

e ≥ 0.3

☑ Greater than

Figure 5.2: *requirements input, belt drive case*

Solutions presentation In the Solutions view, the user is presented with the solutions for a design problem, by means of a 2D plot or a list. A sketch of an individual solution can be included at the right-hand side. A list of all embodiments is visible and connected to the dots in the diagram. Each dot is an embodiment that can be given a color. In this example, different colors represent different belt materials. Both diagram axes can handle all parameters, simply by selecting them from a drop-down box (currently it depicts “optimal belt speed” vs. “pre-load on axis”).



Name	d1	d2	e	t	w	length	Ben...
Embodiment0030	74.56	223.69	563.18	1.12	45.64	714.54	0.033
Embodiment0031	130.32	390.97	647.32	1.61	111.13	911.19	0.033
Embodiment0032	60.56	181.67	832.07	1.55	17.33	956.28	0.033
Embodiment0033	79.54	238.62	539.04	1.25	79.05	700.61	0.033
Embodiment0034	110.51	331.53	623.34	1.91	79.57	848.17	0.033
Embodiment0035	79.61	238.82	759.91	1.59	57.96	922.3	0.033
Embodiment0036	154.91	464.74	614.51	1.86	145.52	928.05	0.033
Embodiment0037	86.23	258.69	335.47	1.26	22.01	510.45	0.033
Embodiment0038	100.96	302.89	511.37	2.21	81.5	717.71	0.033
Embodiment0039	79.07	237.22	431.17	1.62	54.96	592.57	0.033
Embodiment0040	136.47	409.42	400.4	2.05	97.66	677.44	0.033
Embodiment0041	118.29	354.88	421.64	1.58	61.95	661.39	0.033

Parameter 'e' was changed

Figure 5.3: *solutions representation, belt drive case*

Chapter 6

Implementation and realization

The knowledge engineering method of Chapter 4 and the software development method of Chapter 5 are applied to two groups of cases: tacit industrial and explicitly documented.

Two tacit industrial cases are done in cooperation with two companies from Dutch industry. The first case discusses the software development for a product component. The second case concerns the layout design of a transport network. Both cases did not have explicitly documented knowledge before hand.

The second group of cases are machine elements with explicitly documented knowledge. Prototypes are developed to demonstrate PaRC for more familiar designs.

The generic software framework as described in Section 5.4 is developed in C#.NET, with the functionality of the graphical user interface, synthesis and analysis. The software architecture and several class diagrams are briefly discussed to shed some light on the implementation of the software. The prototypes are developed using the generic framework.

6.1 Architecture

The software architecture is developed with the paradigms as discussed in Section 5.4. The central data storage and separation of the synthesis and analysis module are the most visible properties of the architecture, depicted in Figure 6.1. The case specific software modules are the PaRC model, analysis module and some graphics in the user interface.

The user interacts with the software through an input and output interface. The input module gathers the requirements and sends this to a central data storage module. The central data storage handles the data administration and coordinates the activities. The synthesis module receives the requirements as input to generate an embodiment. The algorithm communicates with a PaRC model through a generic interface. A case specific analysis method calculates the performance parameters, again through a generic interface.

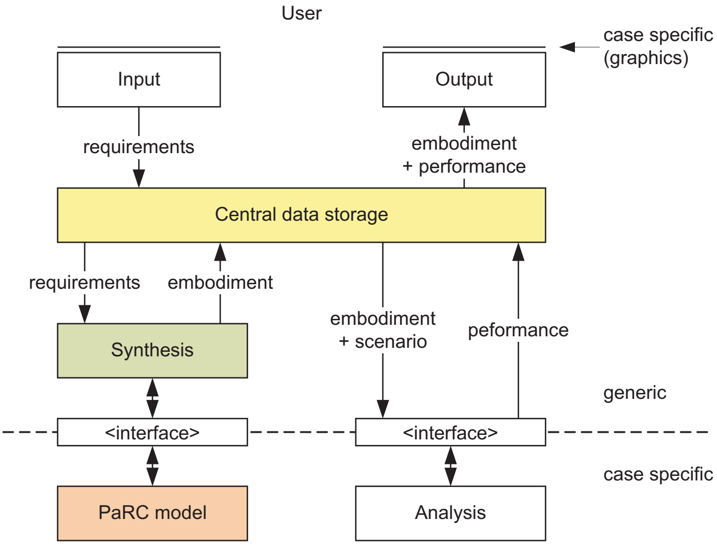


Figure 6.1: *architecture*

The central data storage has a class structure where a design project is divided in multiple sessions, Figure 6.2. Each session is a design problem that begins with requirements and receives multiple embodiments. The requirements are specified for element types, which means that element instances of the same element type have identical requirements.

The synthesis module contains classes to store the generation process and the end result, Figure 6.3. A single root element is the starting point of an embodiment. An embodiment can contain multiple elements, each of which has its own solution state to store the parameter values. The element tree grows as the synthesis process advances. Each expansion of the embodiment is stored as transaction, being either an element expansion or parameter resolve action. The backtracking activity, coordinated from the algorithm, uses the transactions to reverse the synthesis process.

PaRC models contain element types, which in turn consist of parameters and knowledge rules, Figure 6.4. The knowledge rules are divided in X-, R- and C-rules. Parameters are described by a type, a value and a solution space. The type (e.g. embodiment) is defined during knowledge engineering, the value and solution space can be user defined (requirements) or generated during synthesis.

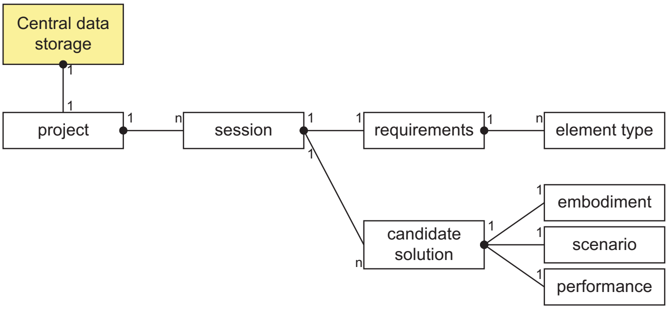


Figure 6.2: *central data storage, class model*

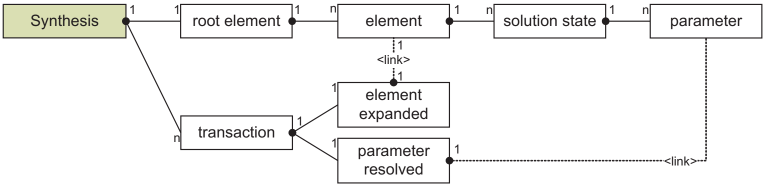


Figure 6.3: *synthesis module, class model*

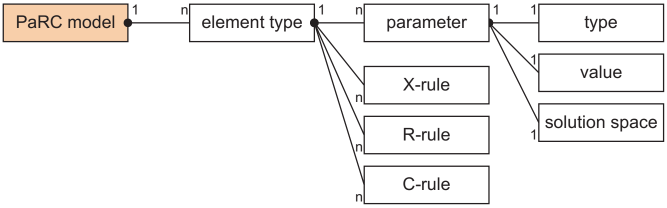
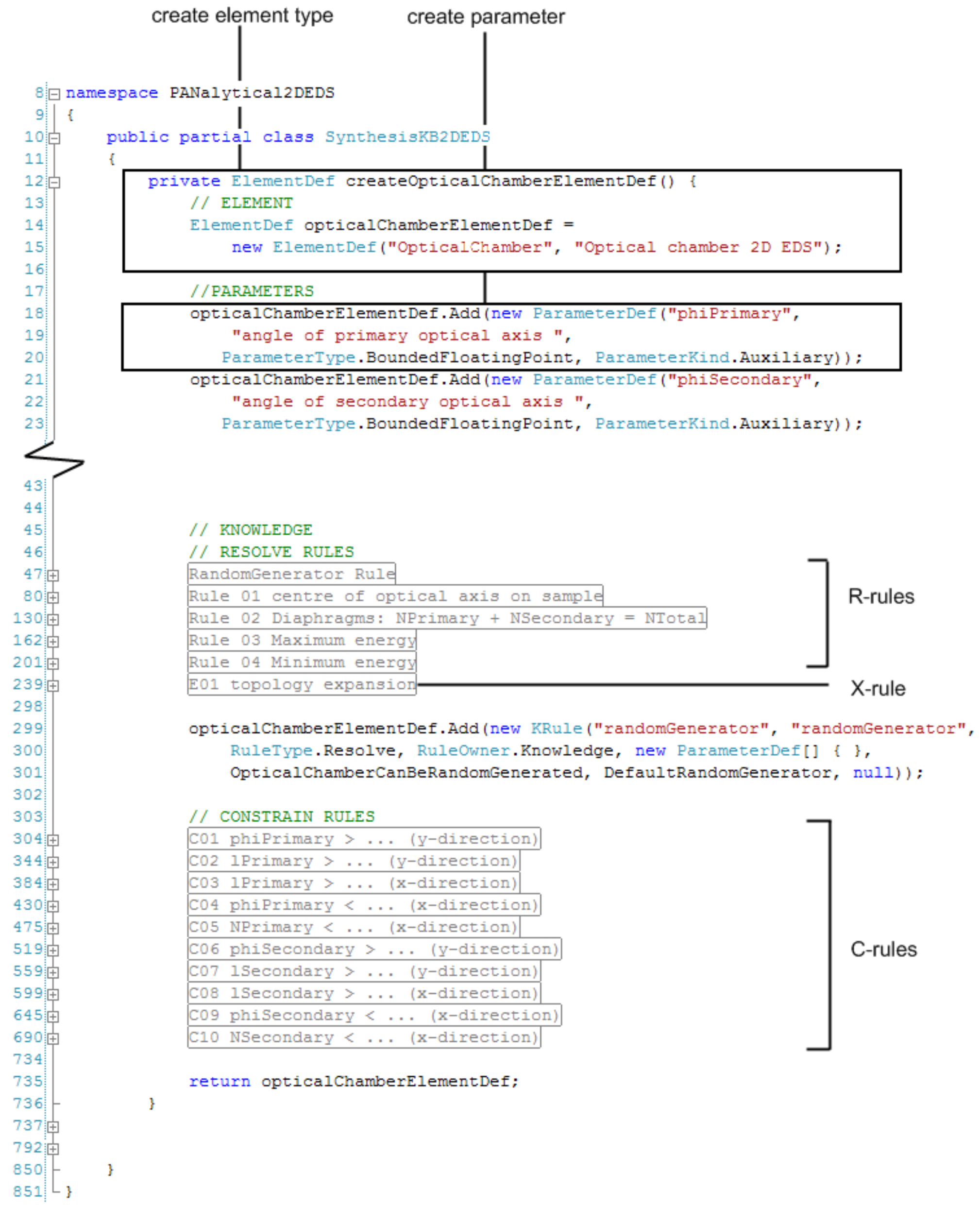


Figure 6.4: *PaRC model, class model*

An implementation example of a knowledge base is given in Figure 6.5. The code illustrates how an element type (in this case the OpticalChamber) is defined in terms of parameters and knowledge rules.

The complete implementation contains 100+ classes with derivatives and overridden methods. The generic software is constantly being modified to test new algorithms. Detailed discussion of the class structures or activity diagrams is outside scope of this thesis.



```
namespace PANalytical2DEDS
{
    public partial class SynthesisKB2DEDS
    {
        private ElementDef createOpticalChamberElementDef() {
            // ELEMENT
            ElementDef opticalChamberElementDef =
                new ElementDef("OpticalChamber", "Optical chamber 2D EDS");

            //PARAMETERS
            opticalChamberElementDef.Add(new ParameterDef("phiPrimary",
                "angle of primary optical axis ",
               ParameterType.BoundedFloatingPoint, ParameterKind.Auxiliary));
            opticalChamberElementDef.Add(new ParameterDef("phiSecondary",
                "angle of secondary optical axis ",
               ParameterType.BoundedFloatingPoint, ParameterKind.Auxiliary));


            // KNOWLEDGE
            // RESOLVE RULES
            RandomGenerator Rule
            Rule 01 centre of optical axis on sample
            Rule 02 Diaphragms: NPrimary + NSecondary = NTotal
            Rule 03 Maximum energy
            Rule 04 Minimum energy
            E01 topology expansion

            opticalChamberElementDef.Add(new KRule("randomGenerator", "randomGenerator",
                RuleType.Resolve, RuleOwner.Knowledge, new ParameterDef[] { },
                OpticalChamberCanBeRandomGenerated, DefaultRandomGenerator, null));

            // CONSTRAIN RULES
            C01 phiPrimary > ... (y-direction)
            C02 lPrimary > ... (y-direction)
            C03 lPrimary > ... (x-direction)
            C04 phiPrimary < ... (x-direction)
            C05 NPrimary < ... (x-direction)
            C06 phiSecondary > ... (y-direction)
            C07 lSecondary > ... (y-direction)
            C08 lSecondary > ... (x-direction)
            C09 phiSecondary < ... (x-direction)
            C10 NSecondary < ... (x-direction)

            return opticalChamberElementDef;
        }


    }
}
```

Figure 6.5: *implementation example*

6.2 Industrial cases

This section discusses two industrial cases: a product component and a network layout.

6.2.1 Optical chamber of an XRF spectrometer

This case is done together with PANalytical and deals with the design of an optical chamber of an x-ray spectrometer. The order of steps from Chapter 5 is followed.

Overview and selection

PANalytical is a high-tech company in The Netherlands that designs and manufactures, among others, x-ray fluorescence spectrometers (Figure 6.6(a)). These instruments are used to determine the chemical composition of materials. The working principle is that of x-ray fluorescence: high-energy x-ray is radiated on a sample material, causing element-characteristic photons to be expelled. These photons are collected by a (solid state) energy dispersive detector and used to determine the chemical composition of the sample, both qualitatively and quantitatively. Wide applications in industry and research are a result of the ability to analyze materials accurately and reproducibly.

The case focuses on the optical chamber of the so-called 2 Dimensional, Energy Dispersive X-Ray Fluorescence (2D EDS XRF) instrument. This case is chosen because of the well-known nature of the physics knowledge and the relative novelty of the product range. A product tree with module specifications is made and revealed that this type of optical chamber is exclusively designed for this type of product.

Figure 6.6(a) shows the instrument with twelve cups in the carousel, each containing a different sample material. The measurement is executed by positioning a cup above the optical chamber that radiates the sample from underneath. The main components of the optical chamber are schematically depicted in Figure 6.6(b): the x-ray tube radiates the sample, causing fluorescence inside the material. This radiation enters the detector, after which digital data processing constructs the chemical composition. Diaphragms are inserted to control the path of the radiation, and an impenetrable casing encloses the components to shield the environment (and operator).

The components should be positioned in such a way that the tube radiates the sample brightly, and the detector “sees” only the radiated section of the sample. Unfortunately, the diaphragms and casing cause unwanted fluorescence and reflections that also enter the detector and negatively influences the measurement quality.

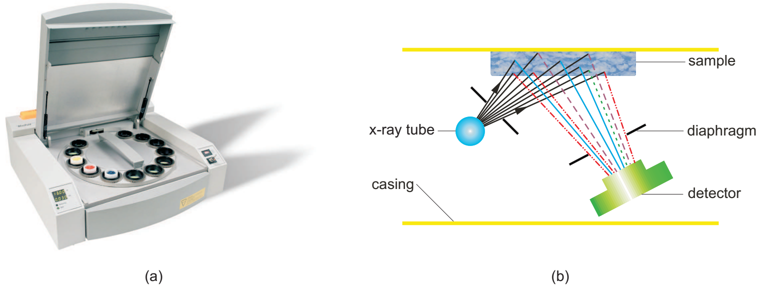


Figure 6.6: *spectrometer and optical chamber (copyright PANalytical BV)*

Modeling

A model of the optical chamber design process is made, using the method from Chapter 4. The steps are described explicitly to demonstrate the implementation of the method.

Levels of abstraction The performance of the optical chamber is specified in terms of its price, size and measurement quality. The measurement quality is determined by two performance parameters of equal importance: signal-to-background ratio and sensitivity. Signal-to-background ratio describes the *relative* amount of sample radiation that the detector receives, and the sensitivity represents the *absolute* amount.

The performance parameter of measurement quality is located at one level of abstraction. A higher and a lower level of abstraction are identified, organized as follows:

1. system design: determines the product specification and functional requirements for the sub-systems. For the optical chamber this means minimum values for signal-to-background ratio and sensitivity;

2. sub-system design, i.e. the optical chamber: determines the geometry, positions and orientations of the main components, from a physics perspective;

3. mechanical design: integrates the optical chamber design with several other sub-systems and prepares the design for manufacturing.

Selection System design and mechanical design lack a predictable and quantitative design process and suitable analysis methods. The sub-system level has suitable analysis knowledge, is fairly predictable and parametric in nature and has an available knowledge source. The design process is mainly the domain of one expertise with over 50 years of combined experience in the field of physics.

The design process consists, for a significant part, of trying new configurations and improving these until the performance is satisfactory.

The design process of the optical chamber on sub-system level is selected for further modeling.

Analysis formalization As discussed previously, the relevant performance of the optical chamber is mainly specified in terms of signal-to-background ratio and sensitivity.

The performance of a particular design is determined by an expert's eye and several analytical estimations. Little simulation analysis is used during the design process, so the analysis method is a relatively tacit one. Ray tracing software is available, but the required calculation time makes this unsuitable to use during the early phases of the design process.

Studying the analysis method and discussion with the expert reveals two levels of detail as far as physics is concerned. The dominant effects of absorption, reflection and fluorescence are of primary importance, while energy shifts, polarization and penetration depth are of secondary.

A finite element analysis method is developed based on these assumptions to simulate an expert's judgment of the quality of a design. The formalization (and automation) of the analysis method leads to the identification of the embodiment parameters. The scenario is determined on the system level and contains 4 parameters: sample material, the tube material and size and detector size.

Synthesis formalization The analysis method requires an embodiment model with the expressiveness depicted in Figure 6.7: 22 parameters describe the geometry, position and orientation of the components. Discussing the embodiment with the expert, and how this is created, revealed 18 auxiliary parameters to capture the design intent of the x-ray application and several geometric construction parameters. The R- and C-rules are acquired for each embodiment parameter.

The parameters are grouped into 8 element types: optical chamber, tube, detector, sample, diaphragm and casing. In addition, there are two element types that represent the two diaphragm sets, one on either side of the sample. This is done because there is different knowledge for the positioning and design of diaphragms on either side. Only a single diaphragm is depicted, but multiple diaphragms can be inserted.

A full description of the synthesis knowledge is provided in Appendix A.1 and summarized in Table 6.1. An example of a (simplified) X-rule is given in pseudo code. The rule adds a number of diaphragms to the primary set, on a random x-coordinate between the tube and the sample. The diaphragms are placed from left to right, and (in this example) a C-rule checks if the diaphragms do not intersect.

```
// CONDITION
if ((DSetPrim.IsNotExpanded            ) &&
    (OC.GetParameter(Nprim).IsResolved ))
{
   // X-rule can be executed
   belief = 1.0
}


// ACTION: create sub-elements
{
// how many diaphragms?
Nprim = OC.GetParameter(Nprim).Value

// minimum x-coordinate (right side of tube)
min = CalculateRight(
          Tube.GetParameter(x).Value,
          Tube.GetParameter(phi).Value,
          Tube.GetParameter(A).Value)

// maximum x-coordinate (left side of sample)
max = CalculateLeft(
          Sample.GetParameter(x).Value,
          Sample.GetParameter(d).Value)

// add diaphragms
for(i = 1 to Nprim)
{
   // instantiate elements
   element D = new DiaphragmType.Instantiate()

   // generate random x value
   x = random(min, max)

   // resolve x
   D.GetParameter(x).Value = x

   // new limit for minimum x
   min = x

   // add diaphragm
   DSetPrim.Add(D)
}
```

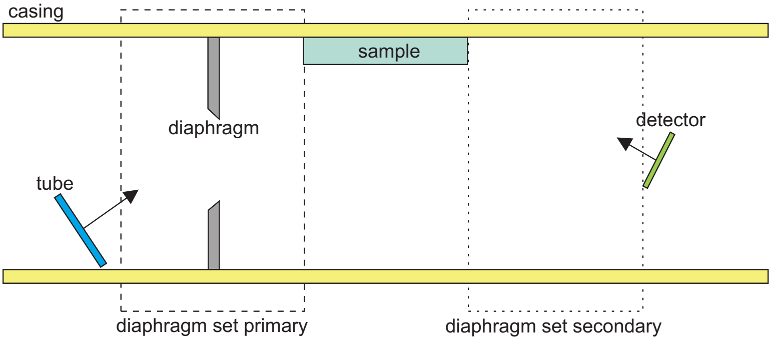


Figure 6.7: *optical chamber, embodiment*

Table 6.1: *optical chamber, synthesis knowledge summary*

PaRC entity	**quantity**
element type	8
parameter	44
X-rule	3
R-rule	19
C-rule	20

Figure 6.8 shows the parameter dependency graph of the synthesis knowledge. The labels in the nodes are formatted as Expression (6.1), using the element names from Table A.1 and parameter names in Table A.2.

$$< \text{element name} > \_ < \text{parameter name} > \tag{6.1}$$

Software development The user interface is shown in Figure 6.9. The insert window shows the required number of embodiments to be generated (200) and the mesh settings for analysis.

Figure 6.10 shows the output of the software and several customized analysis outputs. The top left section displays a plot of the solution space, where each dot is an embodiment (about 2.500 are generated). This particular plot shows signal-to-background plotted against sensitivity, but any combination of parameters can be selected on the axes. The top right corner shows a sketch of the selected embodiment, with directly below it some detailed quantification of the performance. Bottom row shows (right to left): an estimation of the spectrum, an estimation of the detector view (horizontal axis is the length of the detector), and a more detailed view of the background ratio (in this case revealing that the tube is by far the largest contributor).

Validation Validation of the software is done through expert evaluation. Several scenarios and parameter sensitivities are compared to expected values from experience.

The quality of the analysis module is validated as proportional to expectations. This means the software can be used to rank solutions according to calculated performances, which is in accordance with the intended use of a synthesis tool.

Validation of the generated solutions shows expected characteristics, although the “shape” of the solution clouds (e.g. Figure 6.10) is sometimes surprising. The expert whose knowledge is implemented expects a significant reduction in design time.

The knowledge document and the division of the design process into levels of abstraction provide insight in the activities and decision moments. The information between levels is made explicit and aware to the relevant people. This helps to prevent miscommunication and unnecessary iteration loops.

The explicit nature of the shape of the cloud serves as communication tool between disciplines. For experts from different domains, the trends and limitations of solution cloud is helpful to discuss possibilities. Figure 6.11 shows the solutions in a plot of a performance parameter versus an embodiment parameter. From the figure, it is clear that a limit exist for the embodiment parameter to achieve a certain minimum performance. Experts from other disciplines see what it means when they ask a 10% increase of an embodiment parameter.

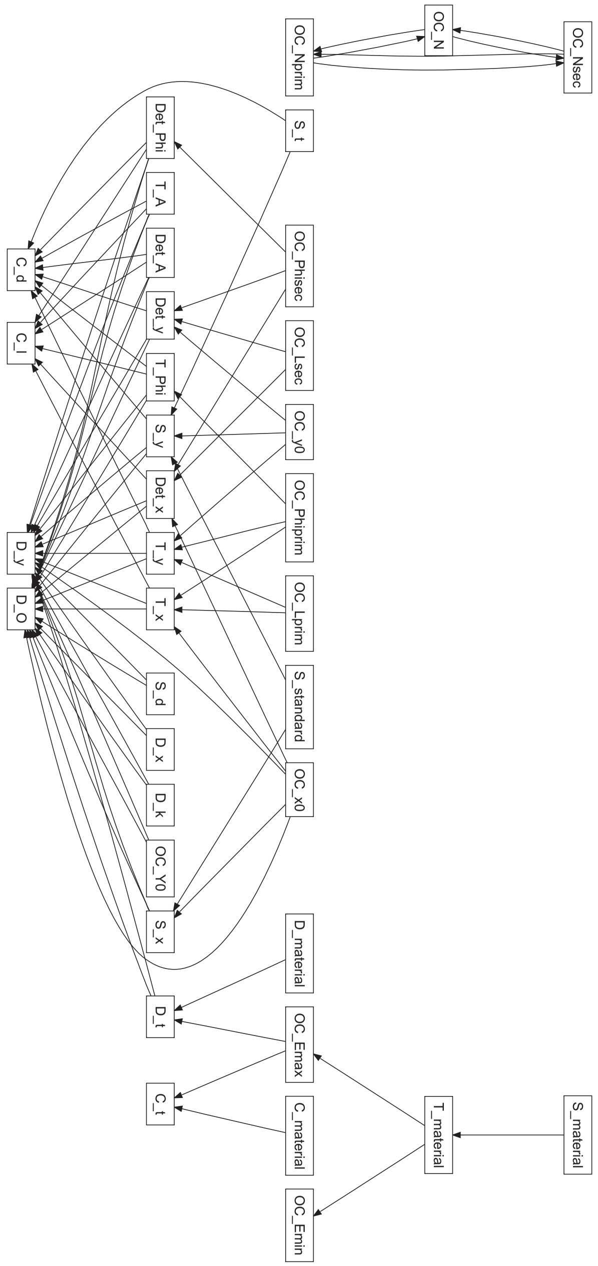
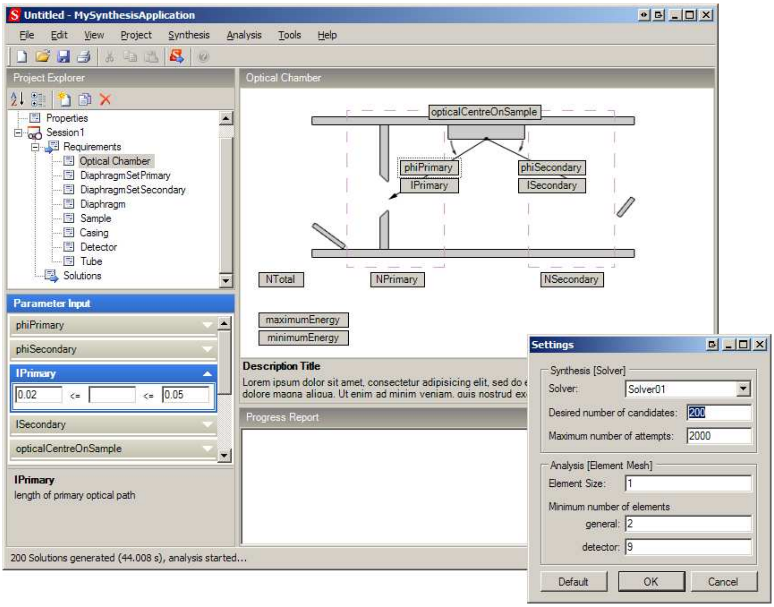


Figure 6.8: *parameter dependency graph, optical chamber*



lPrimary: 0.02 <= [ ] <= 0.05

lPrimary
length of primary optical path

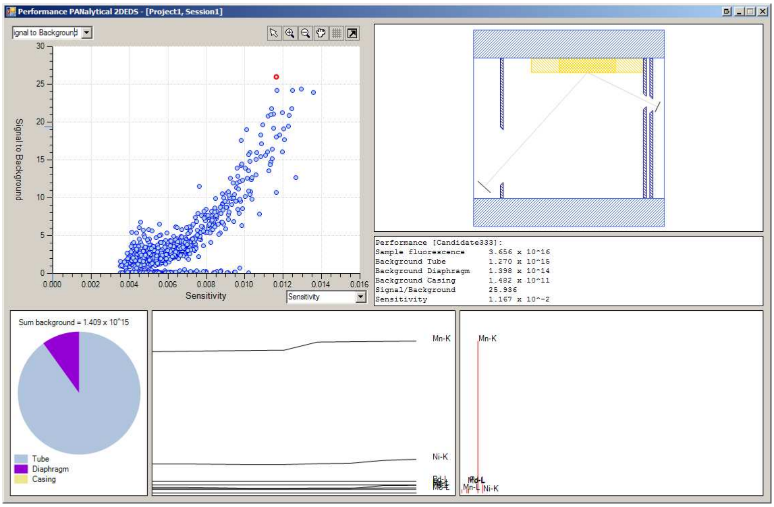
Description Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do e
dolore magna aliqua. Ut enim ad minim veniam, quis nostrud ex

200 Solutions generated (44.008 s), analysis started...

Synthesis [Solver]
Solver: Solver01
Desired number of candidates: 200
Maximum number of attempts: 2000

Analysis [Element Mesh]
Element Size: 1
Minimum number of elements
general: 2
detector: 9

Figure 6.9: *user interface, input*



```
Performance [Candidate333]:
Sample fluorescence      3.656 x 10^16
Background Tube          1.270 x 10^15
Background Diaphragm     1.398 x 10^14
Background Casing        1.482 x 10^11
Signal/Background        25.936
Sensitivity              1.167 x 10^-2
```

Figure 6.10: *user interface, output*

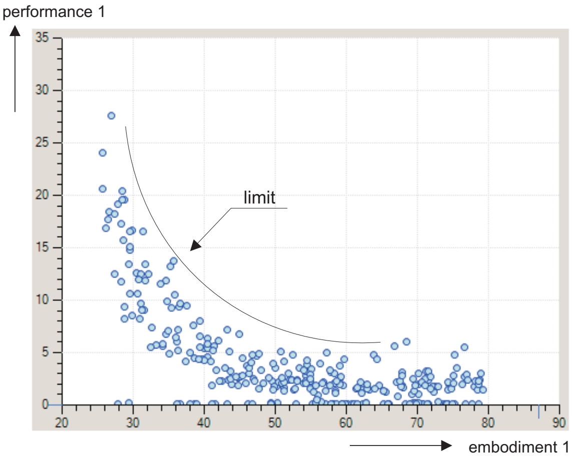


Figure 6.11: *solution cloud*

6.2.2 Baggage handling system

A synthesis module is developed for the design of transport networks, in cooperation with Vanderlande Industries. The topological freedom of such designs is modeled by introducing multiple levels of element types. The knowledge to expand the topological tree is located inside the X-rules.

Overview Vanderlande Industries is a Dutch company that designs and manufactures, among others, baggage handling systems (BHS). These multi-million dollar systems are found in larger airports to process the baggage through a series of belts, scanning devices and sorters toward the aircraft. A schematic view of such a BHS is shown in Figure 6.12.

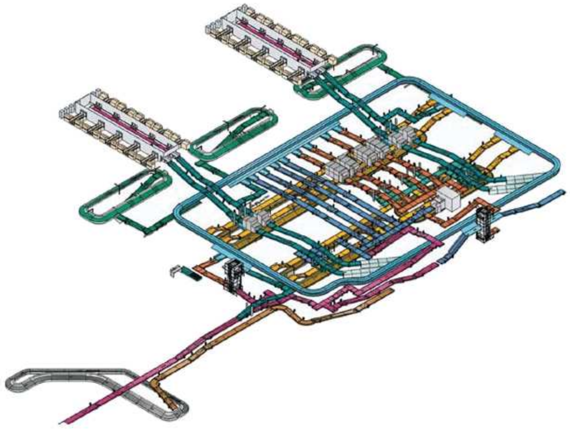


Figure 6.12: *baggage handling system (copyright Vanderlande Industries)*

The BHS design process begins with an airport as customer and ends with technical drawings of all parts. The process is divided into three major levels of abstraction:

1. process flow design (PFD): determines the functional design of a BHS, i.e. what process operations the system performs on a piece of baggage, such as check-in, screening and sorting. A PFD design is shown in Figure 6.13. The design is determined in several discussions with the customer;

2. material flow design (MFD): translates the process flow design into a quantified transport network. At this level there are about 20-30 distinct functional pieces of equipment that constitute a BHS design;

3. detail design: specifies all mechanical and electronic equipment, ready for production.

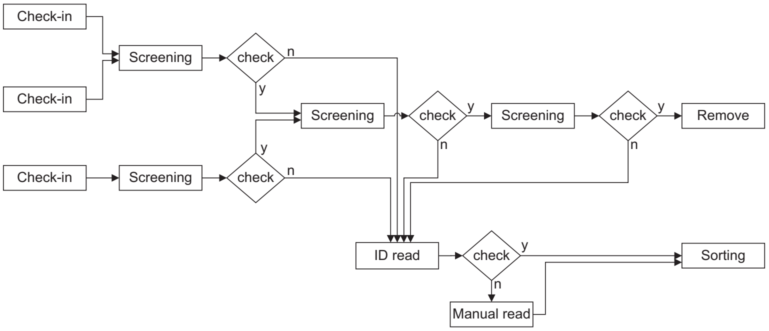


Figure 6.13: *process flow design (PFD)*

Selection The second phase (material flow design) is chosen because it has a predictable design process and quantifiable performance parameters. The input for the design process a process flow design (PFD).

MFD design concerns the arrangement of equipment into a network. The baggage is transported over belts and moves from check-in to exit points. Main design questions are the configuration of equipment into groups and the arrangement of belts between equipment groups.

Modeling the design process Within the MFD level exists two levels of abstraction, each identified by their group of performance parameters. The performance parameters of highest importance are:

1. cost estimation: major pieces of equipment have an estimated cost;
2. capacity: total BHS throughput, in number of bags per hour;
3. redundancy: remaining capacity after failure of a single piece of equipment;
4. in-system time: the maximum time it takes a piece of baggage to travel from check-in to output point.

The secondary performance indicators are measures for control complexity, space efficiency, tracking performances and availability. Only the first group of performance indicators is taken into account.

Analysis Formalization of the analysis methods is done by automating several methods from queuing theory and logistics, discussed in more detail in [15] and [30]. The analysis methods require the embodiment to be a network of pieces of equipment, connected to each other by belts. Because these belts are given an average length, no equipment requires a (three dimensional) position. This reduces the problem size considerably.

The analysis methods led to the embodiment parameters and element types. The PFD processes are high level embodiment element types with a generic structure: illustrated in Figure 6.14. Each PFD process expands with three sub-elements: belt network, equipment group set and baggage flow. The belt network collects the baggage flows from upstream PFD processes. It will construct a network of belts, merge and divert equipment to rearrange the baggage flow. The equipment group contains the pieces of equipment that operate on the baggage flows, e.g. screening and sorting. The baggage flow at the right side of the PFD process is ready to be connected to a downstream PFD processes. This downstream PFD process is instantiated once the current output baggage flow is resolved.

The network connections are made through ID tagging: each instantiated element has a parameter that is the ID of the element it is connected to. These parameters define an embodiment.

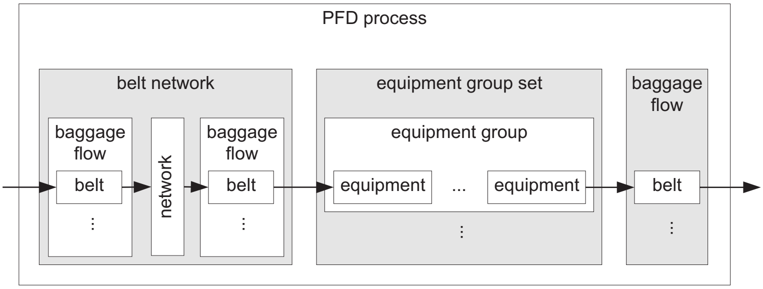


Figure 6.14: *PFD process*

Synthesis Separating the knowledge of different PFD functions breaks the total design problem down into manageable chunks. These PFD functions have an identical structure except the check-in, sorting and baggage removal. The synthesis knowledge is given in Appendix A.2 and summarized in Table 6.2.

Table 6.2: *BHS design, synthesis knowledge summary*

PaRC entities	quantity
element type	51
parameter	204
X-rule	46
R-rule	120
C-rule	0

A representation of a parameter dependency graph for a PFD process is given in Figure 6.15. Parameter dependency graphs are not very well suited to represent knowledge with many topological degrees of freedom. The figure shows the knowledge as if a PFD process would have only one element of each type. The labels in the nodes are formatted as Expression (6.2).

$$< \text{element name} > \_ < \text{parameter name} > \tag{6.2}$$

The top section describes X-rules that construct the topology of a PFD function. The equipment and belt section contain R-rules to determine their individual baggage flows. The lower section determines the total flow through a PFD and the required number of equipment groups.

The graph shows X- and R-rules to be predominantly arranged in the same direction (downward). A few exceptions exist in the “equipment” and “belt” section. Allow me to explain: in a network, equipment can be attached to equipment or a belt, and vice versa. An equipment can resolve its baggage flow only *after* the preceding belt is resolved: represented as an arrow pointing up. A parameter dependency graph of one specific PFD element type would not have this upward pointing arrows. The reason is that the order belt-belt-equipment-... is known and the knowledge rules directed accordingly.

An example of the action parts of an X-rule is given that performs the first topological expansion: the top-most section of Figure 6.15. An example of a R-rule states the resolve method of the suspect baggage flow on a belt.

```
// X-rule ACTION: expand PFD process Screening1
{
// instantiate elements
element BN   = new BeltNetworkType.Instatiate()
element EqGS = new EquipmentGroupSetType.Instatiate()
element BF   = new BaggageFlowType.Instatiate()

// network connections
EqGS.GetParameter(connectedTo).Value = BN.GetParameter(ID).Value
BF.GetParameter(connectedTo).Value   = EqGS.GetParameter(ID).Value

// add elements
Screening1.Add(BN, EqGS, BF)
}
```

```
// R-rule ACTION: retrieve baggage flow of connected equipment
{
// get connected equipment, get flow
connectedTo = Belt.GetParameter(connectedTo).Value

suspect = GetElementByID(connectedTo).
          GetParameter(suspect)

// resolve suspect flow
Belt.GetParameter(suspect).Value = suspect
}
```

Automation The synthesis module is automated to generate embodiments, based on a PFD design. In case of the PFD layout of Figure 6.13, the generated embodiment is depicted in Figure 6.16. The analysis modules are not integrated due to time constraints.

The feedback from industry is encouraging for further development of design automation software to support the design on systems level.

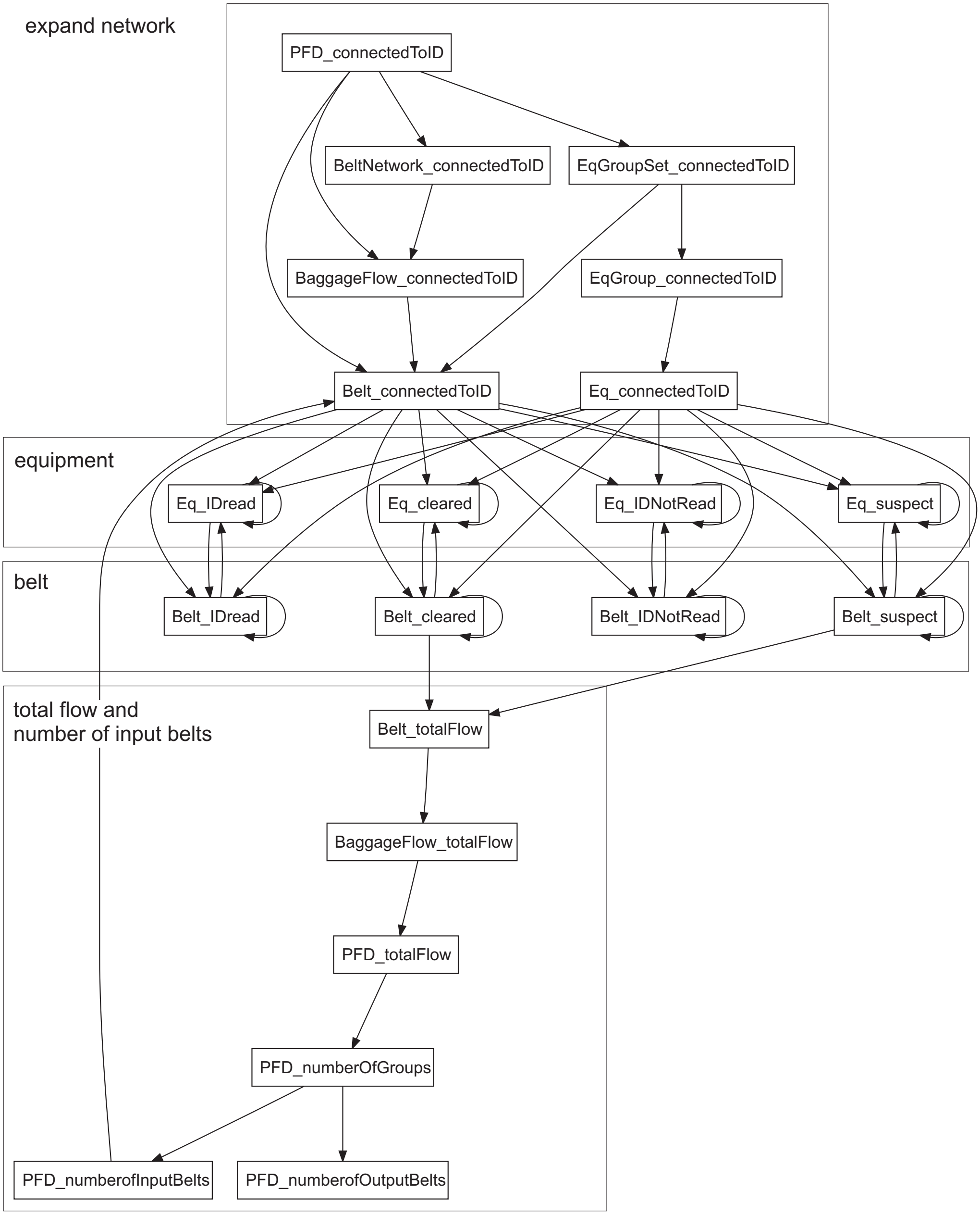


Figure 6.15: *parameter dependency graph, baggage handling system*

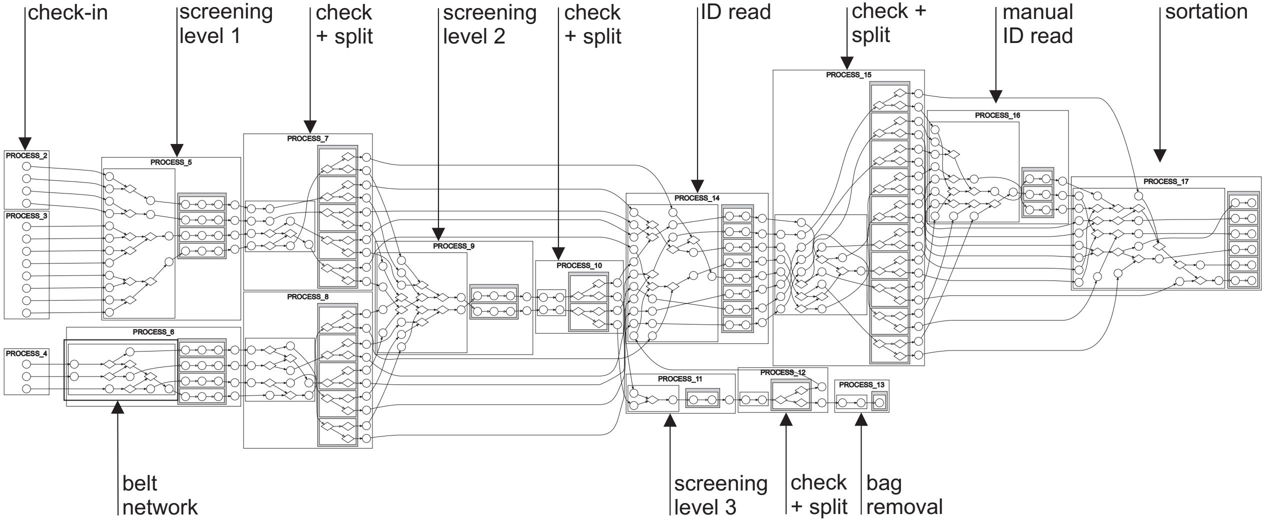


Figure 6.16: *material flow diagram*

6.3 Explicitly documented cases

This section gives PaRC models of four machine elements: a flat belt drive and a compression, extension and torsion spring. The knowledge source is an engineering handbook for design of machine elements [25].

The presented description of machine elements is not meant as *the* knowledge of the element, rather as an example what a PaRC description could look like. The descriptions can be expanded or contracted by taking more parameters or rules into account, or ignoring others. Discussions about the optimal knowledge models of machine elements is outside the scope of this thesis.

6.3.1 Belt drive

A belt drive is a rotating transmission that features a pulling belt as power transmission element, Figure 6.17(a). Knowledge for belt drive design is documented in eight pages [25], including theoretical principles, calculations and several practical considerations for design. An example of how one analysis formula leads to other rules and parameters is given first.

One of the performance parameters is the stress σ within the belt. Quantifying this stress requires some geometric information plus a load scenario: Equation (6.3).

$$\sigma = \frac{F_1}{w \cdot t} \tag{6.3}$$

With F_1 being the force in the pulling part of the belt, and w and t the belt width and thickness, respectively. Following the literature, F_1 is further specified as Equation (6.4).

$$F_1 = F_t \cdot k \tag{6.4}$$

With F_t the force resulting from the applied torque T_1 (6.5) and k being a usability factor derived from embodiment parameters.

$$F_t = \frac{2 \cdot T_1}{d_1} \tag{6.5}$$

The analysis Equation (6.3) leads to a number of parameters: σ, F_1, F_t, T_1, w, t, d_1 and k. Deciding on the type of each parameter is done using the previously discussed decision scheme, leading to:

- embodiment: d_1, w and t;
- scenario: T_1;
- performance: σ, F_1, F_t;
- auxiliary: k.

Including F_1 and F_t in the model increases the expressiveness but can also be discarded and substituted by other parameters. It depends on the goal of the model (e.g. education, automation) which expressiveness is sufficient.

Figures 6.17(b) and (c) show the embodiment and scenario, respectively. The parametric model of a belt drive is stated in Table 6.3.

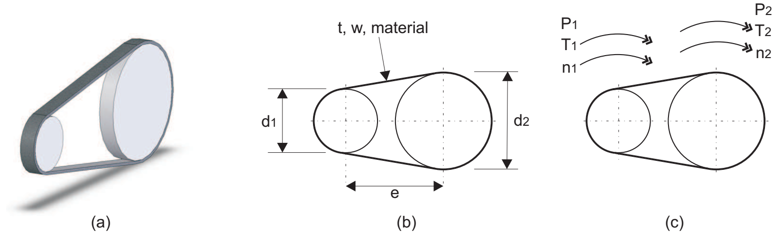


Figure 6.17: *belt drive, embodiment and scenario*

Table 6.3: *belt drive description*

parameter type	name	description
performance	F_a	load on axes
	P_{eff}	effective power transmission
	v	belt speed
	σ_{max}/σ	safety factor on stress
	f_{bmax}/f_b	safety factor on bending
scenario	T_1	torque on pulley 1
	T_2	torque on pulley 2
	n_1	rotation speed of pulley 1
	n_2	rotation speed of pulley 2
	P_1	power on pulley 1
	P_2	power on pulley 2
embodiment	d_1	diameter of driving pulley
	d_2	diameter of driven pulley
	e	distance between axes
	w	belt width
	t	belt thickness
	$material$	belt material
auxiliary	i	transmission ratio
	k	usability characteristic
	m	force ratio
	β	smallest enclosed arc

Continued on next page

Table 6.3 – continued from previous page

parameter type	name	description
	L_{total}	length of construction
	L_{belt}	length of belt
	v_{opt}	optimal belt speed

R-rules The following equations are translated into R-rules, listed in Equation Set (6.6). The right side indicates which parameters can be resolved through the equation.

$$\begin{aligned}
i &= \frac{d_2}{d_1} && \rightarrow i, d_1, d_2 \\
i &= \frac{n_1}{n_2} && \rightarrow i, n_1, n_2 \\
i &= \frac{T_2}{T_1} && \rightarrow i, T_1, T_2 \\
P_1 &= n_1 \cdot T_1 && \rightarrow P_1, n_1, T_1 \\
P_2 &= n_2 \cdot T_2 && \rightarrow P_2, n_2, T_2 \\
P_1 &= P_2 && \rightarrow P_1, P_2 \\
L_{total} &= \frac{d_1}{2} + \frac{d_2}{2} + e && \rightarrow L_{total}, d_1, d_2, e \\
m &= e^{\mu \cdot \beta} && \rightarrow m \\
k &= \frac{m-1}{m} && \rightarrow m, k \\
\beta &= 2 \cdot \cos^{-1}\left(\frac{d_{large} - d_{small}}{e}\right) && \rightarrow \beta \\
L_{belt} &= 2 \cdot e \cdot \sin\left(\frac{\beta}{2}\right) \\
&\quad + \frac{\pi}{2} \cdot (d_{large} - d_{small}) \\
&\quad + \frac{\pi}{2} \cdot \left(1 - \frac{\beta}{\pi}\right) \cdot (d_{large} - d_{small}) && \rightarrow L_{belt}, e
\end{aligned} \tag{6.6}$$

C-rules The C-rules from Equation Set (6.7) are included, in addition to the default solution space for each parameter, such as $d_1 > 0.01$, $d_1 < 0.5$ and a database of 5 belt materials.

$$\begin{aligned}
e &\geq \frac{d_1}{2} + \frac{d_2}{2} && \rightarrow e, d_1, d_2 \\
d_1 &\geq w && \rightarrow d_1, w \\
d_2 &\geq w && \rightarrow d_2, w \\
d_1 &\geq t \cdot \bar{f}_b && \rightarrow d_1, t, material \\
d_2 &\geq t \cdot \bar{f}_b && \rightarrow d_2, t, material \\
w &\geq t && \rightarrow w, t \\
e &\geq 0.7 \cdot (d_1 + d_2) && \rightarrow e, d_1, d_2 \\
e &\leq 2 \cdot (d_1 + d_2) && \rightarrow e, d_1, d_2 \\
if(i \geq 1) : e &\geq 0.9 \cdot d_2 && \rightarrow e, d_2
\end{aligned} \tag{6.7}$$

Where $\bar{f}_b$ is the maximum bending ratio of belt material. The user interface for the embodiment requirements is shown in Figure 6.19, together with the solutions as a plot, list and one sketch. Figure 6.18 shows the parameter dependency graph of a belt drive. Table 6.4 summarizes the knowledge that is taking into account during synthesis.

Table 6.4: *belt drive, synthesis knowledge summary*

PaRC entity	quantity
element type	1
parameter	18
X-rule	0
R-rule	27
C-rule	23

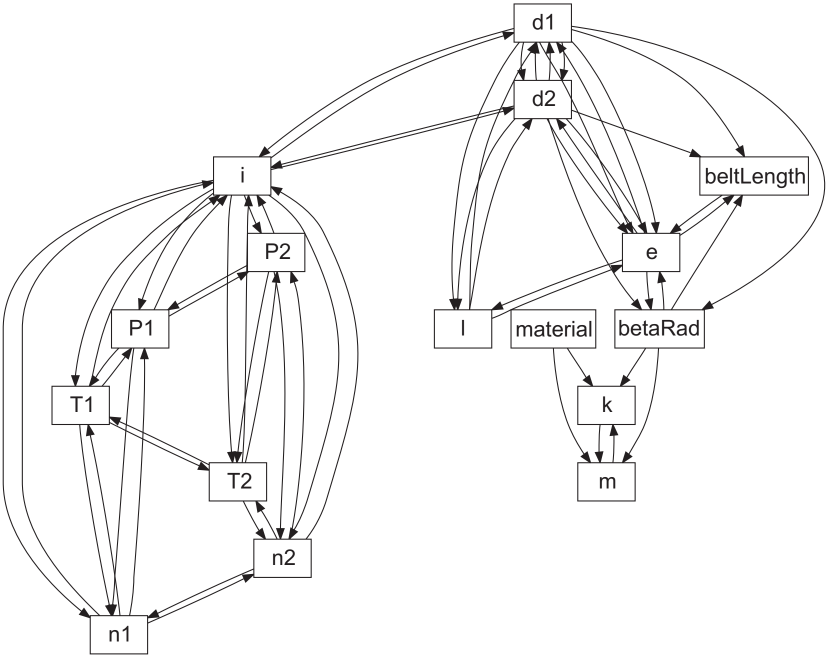


Figure 6.18: *parameter dependency graph, belt drive*

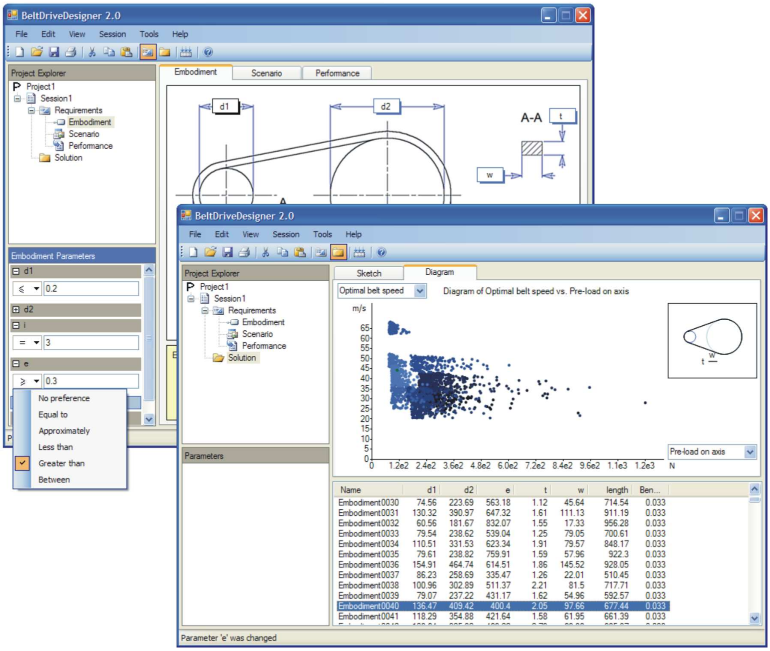


Diagram of Optimal belt speed vs. Pre-load on axis

d1 ⩽ 0.2

i = 3

e ⩾ 0.3

✓ Greater than

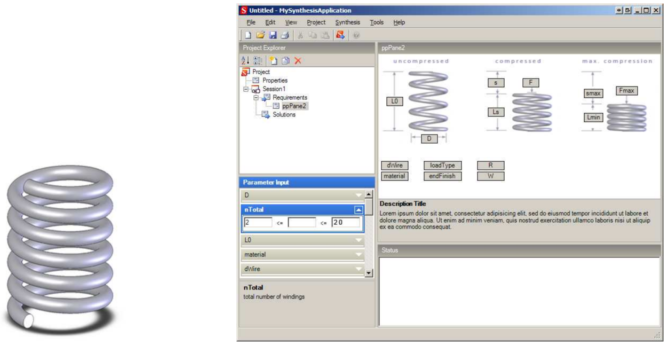
Name	d1	d2	e	t	w	length	Ben...
Embodiment0030	74.56	223.69	563.18	1.12	45.64	714.54	0.033
Embodiment0031	130.32	390.97	647.32	1.61	111.13	911.19	0.033
Embodiment0032	60.56	181.67	832.07	1.55	17.33	956.28	0.033
Embodiment0033	79.54	238.62	539.04	1.25	79.05	700.61	0.033
Embodiment0034	110.51	331.53	623.34	1.91	79.57	848.17	0.033
Embodiment0035	79.61	238.82	759.91	1.59	57.96	922.3	0.033
Embodiment0036	154.91	464.74	614.51	1.86	145.52	928.05	0.033
Embodiment0037	86.23	258.69	335.47	1.26	22.01	510.45	0.033
Embodiment0038	100.96	302.89	511.37	2.21	81.5	717.71	0.033
Embodiment0039	79.07	237.22	431.17	1.62	54.96	592.57	0.033
Embodiment0040	136.47	409.42	400.4	2.05	97.66	677.44	0.033
Embodiment0041	118.29	354.88	421.64	1.58	61.95	661.39	0.033

Parameter 'e' was changed

Figure 6.19: *graphical user interface, belt drive*

6.3.2 Compression spring

Figure 6.20 shows the spring and the GUI of the synthesis tool. Table 6.5 summarizes its synthesis knowledge. A complete description is found in Appendix A.3.



2 <= <= 20

Description Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

nTotal

total number of windings

Figure 6.20: *compression spring*

Table 6.5: *compression spring, synthesis knowledge summary*

PaRC entity	quantity
element type	1
parameter	24
X-rule	0
R-rule	45
C-rule	31

The parameter dependency graph of a compression spring is shown in Figure 6.21.

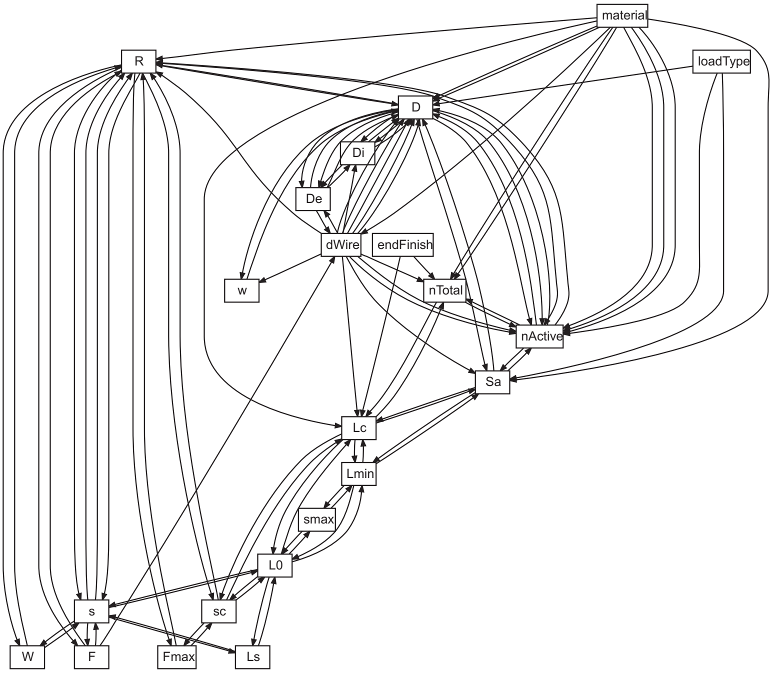
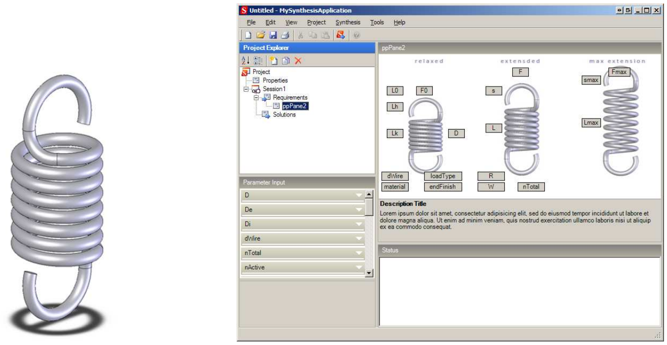


Figure 6.21: *parameter dependency graph, compression spring*

6.3.3 Extension spring

Figure 6.22 shows the extension spring and the GUI of the synthesis tool. Table 6.6 summarizes its synthesis knowledge. A complete description is found in Appendix A.4.



Description Title

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

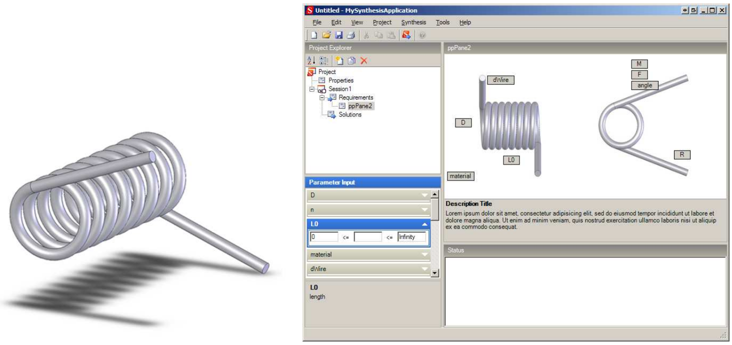
Figure 6.22: *extension spring*

Table 6.6: *extension spring, synthesis knowledge summary*

PaRC entity	quantity
element type	1
parameter	24
X-rule	0
R-rule	42
C-rule	27

6.3.4 Torsion spring

Figure 6.23 shows the spring and the GUI of the synthesis tool. Table 6.7 summarizes its synthesis knowledge. A complete description is found in Appendix A.5.



L0: 0 <= <= Infinity

L0
length

Description Title
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Figure 6.23: *torsion spring*

Table 6.7: *torsion spring, synthesis knowledge summary*

PaRC entity	**quantity**
element type	1
parameter	18
X-rule	0
R-rule	34
C-rule	21

6.4 Comparison

This section compares the models of synthesis knowledge of the previously discussed cases, summarized in Table 6.8.

Table 6.8: *synthesis knowledge, comparison*

case	element types	parameters	R-rules	C-rules	X-rules
belt drive	1	18	27	23	0
compression spring	1	24	45	31	0
extension spring	1	24	42	27	0
torsion spring	1	18	34	21	0
optical chamber	8	44	19	20	3
baggage handl. sys.	51	204	120	0	46

6.4.1 R-rules

Figure 6.24 compares the amount of R-rules with the number of parameters. This indicates the amount of "deterministic" knowledge in relation to the size of the problem. The figure depicts a dotted line that represents equal amounts of parameters and R-rules. It shows that the industrial cases have fewer R-rules per parameter compared to the literature cases. The literature cases are located relatively close together, indicating similar amounts of knowledge for similar problem sizes. The cases of belt drive and the three springs have, on average, more than one R-rule per parameter, a feature that is discussed with more detail next.

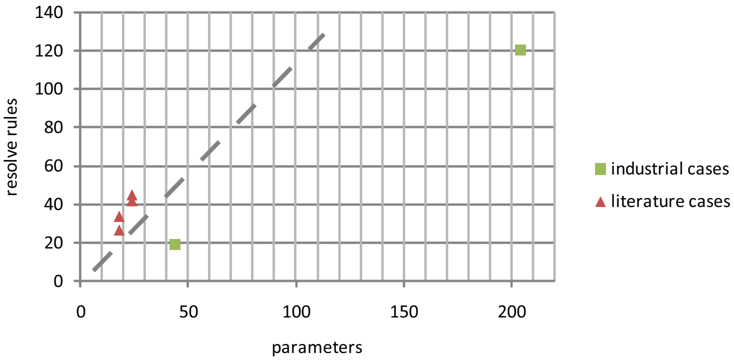


Figure 6.24: *R-rules versus parameters*

The literature cases have an average of 1.8 R-rule per parameter (from Table 6.8: the total number of R-rules divided by the total number of parameters). This is in contrast with the industrial cases, where hardly any parameter has more than one R-rule, as seen in the Appendices A.1 and A.2.

The distribution of parameters over R-rules is given in the histogram in Figure 6.25 and Table 6.9. The histogram shows the number of parameter (y-axis) that can be resolved by 1-7 R-rules (x-axis). For example, the belt drive contains four parameters that have one R-rule each and ten parameter with two R-rules. This means that most parameter can be resolved using multiple rules, and it depends on the available information which rule is used.

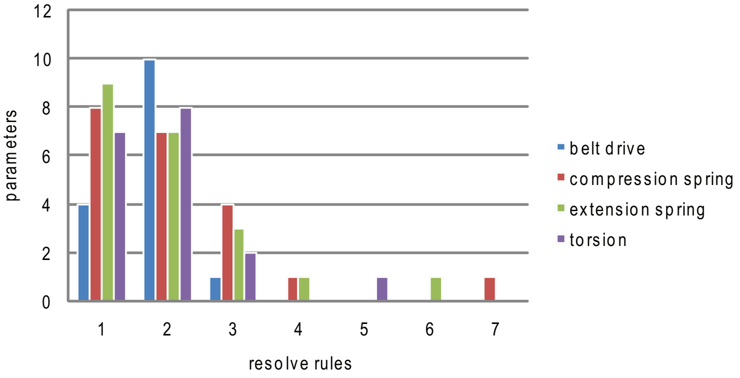


Figure 6.25: *number of R-rules per parameter*

Table 6.9: *number of R-rules per parameter*

case	1 rule	2	3	4	5	6	7
belt drive	4	10	1	0	0	0	0
compression spring	8	7	4	1	0	0	1
extension spring	9	7	3	1	0	1	0
torsion spring	7	8	2	0	1	0	0

6.4.2 Parameter dependency graphs

The parameter dependency graphs also show multiple resolve paths through the parameter network. Figure 6.26 depicts the parameter dependency graphs of the optical chamber and the compression spring, Figures 6.26a and 6.26b respectively. The experience-based knowledge of optical chamber design has several different characteristics compared to the more mathematically documented knowledge of a compression spring.

Expert knowledge from optical chamber design is oriented toward the parameters at the bottom row. The parameters form horizontal groups, or levels, with the R-rules bridging from top to bottom. The mathematical knowledge depicts a more tangled network with arrows going up, down, left and right.

The expert cases show that the order in which the parameters are resolved is relatively fixed. This also means that the set of possible input parameters is limited, because there are no R-rules to traverse or move upward in the network.

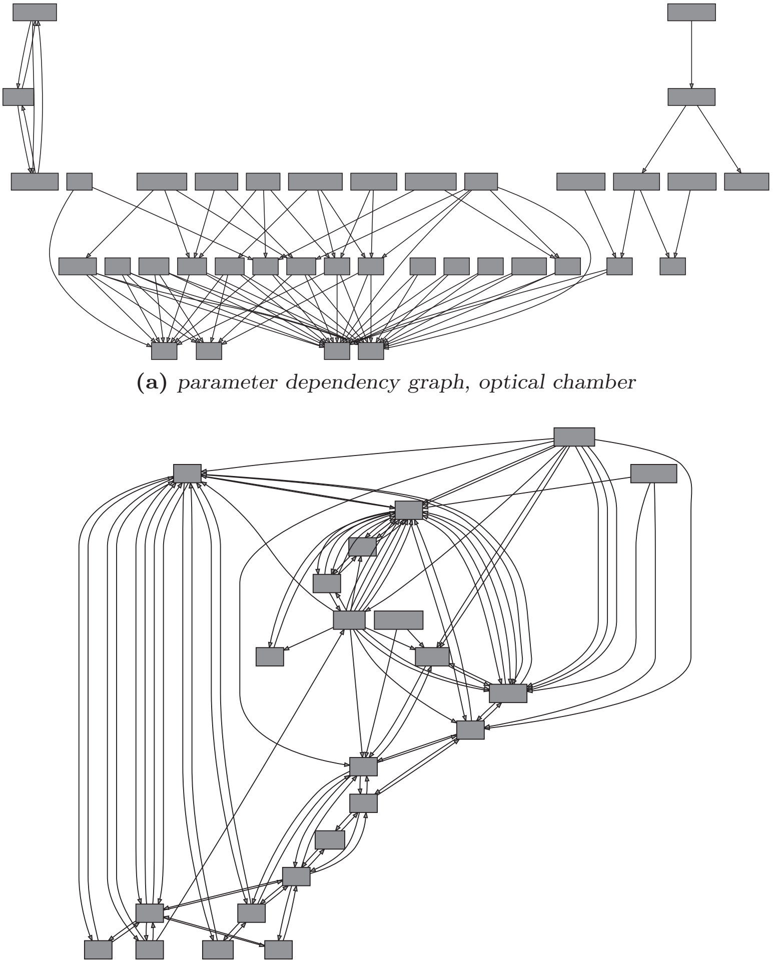
The parameter dependency graph of the spring case has much less orientation in its R-rules. The order in which parameters are resolved is more flexible and depends on the initially known parameter values. The knowledge model has R-rules to move more freely through the parameter dependency graph.

In both cases, the synthesis algorithm decides which rules are executed. The decision is based on the available parameter values, without an explicit strategy stated beforehand. As discussed in Section 3.4, this behavior is termed “opportunistic” in cognitive design research [52].

The industrial cases allow less opportunistic freedom for the algorithm, compared to the literature cases. The order in which “industrial” parameters are resolved is implicitly determined by the R-rules. The algorithm is still opportunistic, it just finds more or less the same path again and again. There would be little difference if the solving strategy from the expert would be explicitly implemented as well.

Comparing the knowledge from industrial and literature cases reveals that industrial (expert) knowledge is more goal oriented toward a fully defined embodiment. As a result, it is less flexible compared to the literature cases.

A characteristic that describes the flexibility, the routine-like nature and multiple allowed input sets, is the average number of resolve rules per parameter (see Figure 6.24). The industrial cases have less than one R-rule per parameter. Both industrial cases have relatively fixed input sets, an implicit strategy for synthesis and fewer flexibility during the path from requirements to embodiment. More than one rule per parameter indicates flexibility in input set, solving strategy and paths from requirements to embodiment.



(a) *parameter dependency graph, optical chamber*

(b) *parameter dependency graph, compression spring*

Figure 6.26: *comparison of parameter dependency graphs*

6.4.3 Development time

Table 6.10 gives the development times of the synthesis modules. This includes the activities of knowledge acquisition, modeling in PaRC entities and implementation into the generic framework. The time indications are rough estimates without significant scientific value due to the uncontrolled experiment setting, and should be interpreted with 20-30% error margins. It does however give an impression about the development effort that is needed.

The three spring cases show a decreasing value due to the learning effect and reusing of code. Ignoring this effect, it shows that knowledge models of similar size require roughly the same development effort. The time for the baggage handling case is double of the optical chamber case, while it has almost five times as many parameters. The reason for the relatively low development time of the BHS case is the level of similarity between the knowledge models of the elements, leading to fast software development.

Table 6.10: *development time for synthesis module*

case	development time (weeks)
belt drive	0.8
compression spring	0.8
extension spring	0.6
torsion spring	0.4
optical chamber	4
baggage handling system	8

Figure 6.27 shows the development times of the cases in relation to those of the early prototypes discussed in Section 2.10. In general, the development effort is significantly less compared to the prototypes, as well as less scattered. The two industrial cases indicate scalability, but stating any firm conclusions requires controlled experiments and a more elaborate data set.

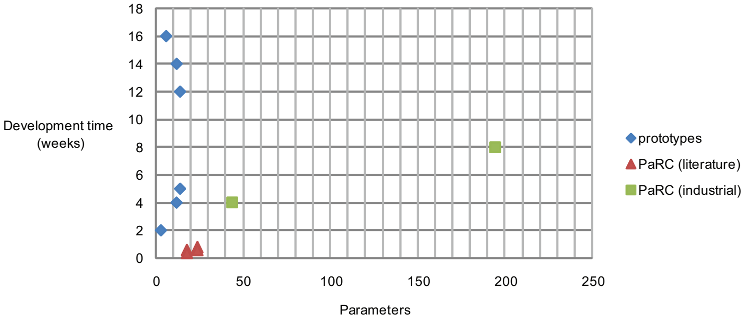


Figure 6.27: *synthesis module development time*

Chapter 7

Conclusions & Recommendations

7.1 Conclusions

Mass-deployment of design automation applications in contemporary industry is not prevented by the *automation* of models, but by the *building* of models from expert knowledge. The tacit nature and the diversity of the knowledge calls for a method and modeling standard that is independent of the design case at hand. This thesis proposes such a method for the scope of parametric engineering design.

This thesis provides a methodological translation of tacit and experience-based knowledge into a model that is subsequently automated to generate multiple design solutions. The development process of the software is prescribed to reduce development effort and enable a modular software architecture.

The activities of the knowledge engineering method are prescribed independently of the design case *and* the automation algorithm. Knowledge engineering becomes an activity without the need to become a design expert or algorithmic expert. Interview-style knowledge acquisition yields answers that are directly usable to develop the model, and a generic algorithm is ready to process any model that meets the standard.

A key issue for efficient knowledge engineering and software development is the consistency between knowledge acquisition and eventual automation: do not acquire something the algorithm cannot process. Consistency is guaranteed by prescribing the activities in terms of a single model of the design process and synthesis knowledge, called PaRC. The model is used to standardize the input and output of the knowledge engineering and software development activities. Standardization enables use of the same methods, procedures and algorithms within the applicability scope.

The contribution to literature lies in the methodological documentation of knowledge engineering of expert knowledge, and subsequent software development for design automation. This thesis further aims to reduce the overall development effort of the software program and make the process more predictable in terms of the activities to execute and the eventual functionality of the software. Within scope, the method is generically applicable and scalable toward design problems of higher complexity in terms of number of parameters and topological complexity.

The overall development time of the design automation software is significantly decreased while the design process itself is made explicit and modeled in a standardized format. The benefits are most significant toward parametric problems with predictable topologies.

Lifecycle aspects of the software are positively affected by the modularity of the software architecture *and* the knowledge model. The architecture modules communicate through generic interfaces that enable exchanging modules without affecting the entire software. The knowledge model consists of self sustaining modules (elements) that contain a part of the product model together with the relevant rules, independent of the algorithm. Modifications to the model are local and do not require algorithmic changes.

All development activities are prescribed in terms of the models of the design process and synthesis knowledge, resulting in several advantages during the six development phases of design automation software:

1. decomposition: a design problem is first broken down into levels of abstraction to discriminate between parameters of higher and lower importance. Software that aims to support engineering design and fit the original design process must acknowledge the division between issues of major and minor importance.

   Decomposing a design process into distinct layers of abstraction reduces the complexity and provides clear system boundaries for the modeling activities. A design problem has been cut down to manageable size because the experts have to do it themselves. Experience led him/her to certain (tacit) models. Missing a model boundary not only leads to a larger problem that does not fit the original context, but the knowledge to solve the problem might be missing because the expert does it differently.

   A second decomposition step models the design process within a single layer of abstraction. The model is divided into sub-processes and sub-sets of information. This phase further reduces complexity and provides system boundaries on module level.

   The model of the design process is determined predominantly by the analysis method, and the analysis method is determined by the expert. The indirect modeling method enables the expert to focus on his/her expertise, while the knowledge engineer receives the necessary information;

2. knowledge acquisition: the begin point of the knowledge acquisition phase is an incomplete PaRC model, and the goal of this activity is to complete the model. The knowledge engineer knows what to look for, because the PaRC modeling entities are known in advance. The explicit nature of the entities improves communication between knowledge engineer and expert, making this phase efficient and fast.

   Both industrial cases show relatively predictable content of knowledge rules. This indicates that experience relates to the model *size*, rather than the rules;

3. modeling: a total of five standardized entities are used to model synthesis knowledge as a PaRC model. Using standardized modeling entities enables efficient modular knowledge engineering of expert knowledge. The resulting models are easily modified and coupled, thanks to the modularity.

   A verbal representation of the knowledge models serves as a knowledge capturing document. The model describes the *conclusion* of experience: the model sizes and knowledge on the "how to move on"-aspect of design synthesis. Directed graphs are used to graphically depict implicit design strategies and flexibility of the problem solving capabilities;

4. automation: PaRC models of synthesis knowledge are automated with a single algorithm. This removes the need for a custom algorithm for each knowledge model.

   Secondly, the algorithm provides the core functionality of the software, which means the characteristics (and limitations) are known almost immediately after modeling, but *before* software development. The design experts see the end goal and its added value before implementation is started. This enables discussions and modifications when it is still easy to change.

   The knowledge is organized in an agent-like manner. Agents in this sense are the PaRC entities of parameter and elements, who manage their own value generation and validity thereof. The supervising algorithm guides the population toward the endpoint: an embodiment. Solution generation is done in a bottom-up manner: the modeling entities manage their own micro-knowledge (X-, R- and C-rules) to create a macro-solution (embodiment);

5. generic software development: the software architecture is made to mimic the model of the design process. The interfaces between the sub-processes are made generic to allow modular software development. Modularity enables a toolkit-like software development process that reduces error sensitivity and required development effort for multiple cases;

6. user interface: the embodiment parameters are varied to explore design alternatives in the solution space. The performance parameters are used as quality measures for parameter sensitivity. Higher level understanding of

the design possibilities and limitations is provided by visualizing the trends and shapes of the “clouds” of multiple solutions. Communication between domains is positively affected by visual presentations of solution clouds. The experts that use the software gain quick insight and use this as a starting point for further optimization, significantly reducing the total design time to reach a sufficiently good design.

The development method is applied to six design cases: two industrial cases with a design expert, and four cases from an engineering handbook. Prototypes are developed for all cases.

The two industrial cases have distinct levels of abstractions that significantly reduce the size of the model, compared to the situation where no levels of abstractions would be used. The resulting software generates design solutions that meet expectations of the design experts. If a completed software system is available, the design process is expected to be strongly reduced, with better insight into the available solutions and improved communication with other domains.

The four cases from the handbook are made into prototypes with significantly reduced development effort compared to the prototypes that are developed without a prescriptive method.

The main difference between knowledge from literature and human expert is the amount of deterministic knowledge (R-rules) per parameter. Literature knowledge has, on average, more ways to calculate a value for a parameter, compared to expert knowledge. This makes literature knowledge more flexible for problem solving while the experts operate in a more “routine” problem setting: the input is relatively predictable and the goal of the design process is known.

7.2 Recommendations

A development method is described as an integration of a number of domains, guided by a central model of synthesis knowledge. First, several recommendations for research directions of the individual domains are given to improve the development of design automation applications, depicted in Figure 7.1.

General recommendations to improve upon the "integrated domain"-approach for software development are given afterward.

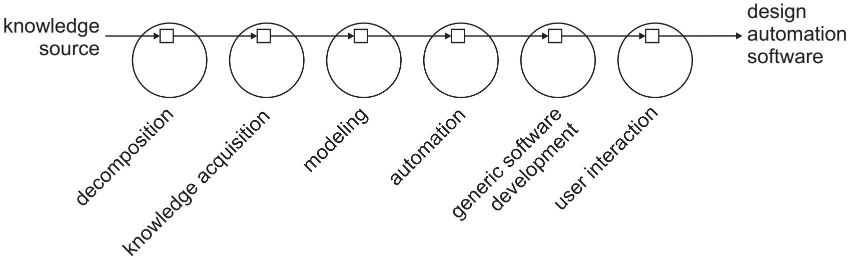


Figure 7.1: *knowledge domains and software development*

7.2.1 Industrial decomposition

The applicability scope is widened if a broader range of design processes can be decomposed and modeled. Examples are manufacturing related design processes and conceptual design. The use of static concepts (such as an analysis method) are found helpful as a starting point to prescribe the decomposition activity. For manufacturing related processes, it seems interesting to start from the manufacturing process to decompose the relevant information.

Conceptual design can be seen as a construction of conceptual form-function entities, with the connections as (topological) degrees of freedom. Graph grammars are used to generate conceptual designs using grammars as form-function entities [23]. A generic layout of a function entity can be used as static entity to decompose such design processes.

7.2.2 Knowledge acquisition

A wide range of interviewing techniques exists. The method of this thesis offers a clear begin point and end goal. Existing technique can be used to increase efficiency and also document knowledge that is not needed for pure automation. If the core-knowledge is the knowledge required for automation of the design task, the peripheral knowledge can explain why and relate to researches, studies or important previous experiences.

7.2.3 Modeling and automation

Challenges on modeling remain for shapes, conceptual design and systems of equations, among others. Fortunately, a large body of research is available on the modeling and automation of (design) problems. The proposed PaRC model can be improved by adding existing algorithms, such as self-learning, look-ahead and more intelligent conflict solving, strategy planning and optimization.

The knowledge itself can be subjected to evaluation, making it possible to discuss the correctness of individual knowledge rules. The software can provide feedback on the knowledge base by showing statistics about the frequency a rule is used, or how often a parameter has a conflict. This insight into knowledge can help guide research activities to improve rules or discover new knowledge.

Individual PaRC models can be connected as a flexible network of agents because of the already agent-like organization of elements and parameters. An X-rule can connect to an element, which is a software program on its own. The interface between such systems is the set of requirements, after which the "sub-program" can begin its own thread of activities as if it were a normal element. This enables a flexible network of agents, as suggested in [47].

Filtering between PaRC models is essential to prevent explosion of design alternatives. The performance parameters are logical choices to constitute the objective functions, and only a certain percentage of solutions is selected to proceed to the next phase.

Existing optimization methods are one of the first expansions in algorithmic capabilities of the software that will add value for the user. Knowing the Pareto fronts in multi-objective optimization gives insight in the relevant outlines of solution spaces, especially if many objectives exist.

Including optimization knowledge to PaRC could be done by rules that use the same structure as R- and X- rules: O-rules. Such O-rules have the additional condition that analysis is executed previously. Their action will propose a (hopefully) better value for the embodiment parameters or topological structures. The algorithmic process of adjustment behaves similar to synthesis, with the difference that parameter values have a "preferred" value, given by the O-rule. The generative algorithm takes the advice of the optimization rules into account when deciding upon the next step. The parameter or element that is addressed tries to satisfy the advised value, if the constrains allow it.

7.2.4 Generic software development

Software development effort is reduced if a generic toolkit, or development kit, would be available that contains ready-to-use algorithms, which all have generic interfaces to the modeling entities [48]. If an open-source toolkit is available for a standardized knowledge language, the functionality of the software can be increased almost without end.

The Model-Driven Architecture (MDA) approach offers a promising research direction to further reduce (software) development cost and improve quality for a truly mass-deployment of software on different programming languages and platforms. Technologies and techniques for model-driven software generation are developed such as meta modeling, language definition and extension mechanisms (such as UML). If the MDA research efforts are combined with standardized knowledge modeling, the boundary between generic code and specific code is further advanced. A domain specific modeling language (to define the knowledge models) is the starting point to develop software programs independently of the application's platform.

7.2.5 User interaction

An intuitive and rewarding interaction between user and software is critical for acceptance in industry. Many things can be done once a cloud of solutions is generated. For instance: identification of the Pareto fronts, interactive clouds where different experts can select and de-select solutions. Multiple cross-sections of the solution space can be placed next to each other, and the performance of a single solution is seen in all clouds simultaneously. Experts can ask for specific sensitivity studies, or optimization toward specific corners of the solution space by relaxing certain requirements.

Improvement of user interaction can also take the direction of interactive knowledge bases. Users manage their knowledge bases without coding: the domain of knowledge management is included into the development procedure.

7.2.6 General

A number of general recommendations for integrated software development methods are suggested in figure 7.2, depicted as options A, B and C.

The first possibility (A) suggest the inclusion of other domains in the development process. For instance, Virtual Reality techniques can be included to offer new user interaction possibilities. Theories and models from cognitive design science can be helpful when following the same knowledge-guided approach as done in this thesis.

B signifies the use of alternative concepts within a domain, in a flexible manner. For example, a range of alternative automation and optimization algorithms can be prepared and plugged in, depending on properties of the problem. This leads to a toolkit-like approach with generic software modules, greatly advancing the computational power of the software that can be developed.

C suggest the development of a similar development method for other software functionalities, such as pure optimization or conceptual design. Different domains might be involved, but the approach to use a knowledge model as central concept can be used.

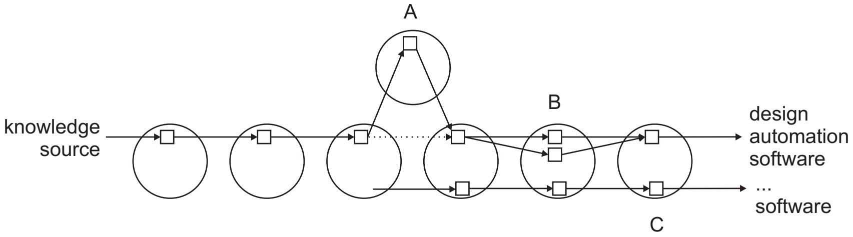


Figure 7.2: *alternative development procedures*

Acknowledgement

This thesis is part of the research project “Smart Synthesis Tools”, started in 2005. The author gratefully acknowledges the support of the Dutch Innovation Oriented Research Program “Integrated Product Creation and Realization (IOP-IPCR)” of the Dutch Ministry of Economic Affairs.

Professor Fred van Houten, dank voor onze leuke en nuttige discussies. Maar ook voor de vrijheid die ik gekregen heb om de Wondere Wereld der Wetenschap te mogen verkennen (vooral Las Vegas! En Kroatië was ook goed...).

I would also like to thank Professor Tomiyama and Professor Shea for the interesting discussions about synthesis, science and design. Professor Wengenroth and Professor Bauchau for their excellent presentations at the right time. I thank you all for giving me critical input and feedback during my research.

Tevens ben ik mijn dank verschuldigd aan alle heren uit de industrie die een bijdrage hebben geleverd aan mijn promotie. Met name mijn begeleiders bij PANalytical en Vanderlande: Walter van den Hoogenhof, Rob de Lange en Roy van Putten. Daarnaast dank ik ook de rest van de begeleidingscommissie voor de discussies en feedback.

Furthermore, I would like to give some more personal thanks to a number people “without whom, none of this would have been possible”. First and foremost my parents, op wie de voorgaande frase in de meest letterlijke zin van toepassing is. Jullie steun en gezelschap is enorm belangrijk voor mij en heeft mij veel geholpen. Ik dank jullie, en hoop er nog lang van te kunnen genieten. Ik hoop dat het nieuw huisje het warme hart van de familie wordt.

Kelly, mijn lieve vrouw, die aan mijn zijde stond gedurende de afgelopen vier jaar. Die al dat geneuzel over kennisregels heeft moeten aanhoren. Respect, liefde en steun. Zonder jou...

Zus Maaike, die op enige afstand toch altijd dichtbij is. Hopelijk tot snel! Zusjes Annemiek en Leontien die beide ontzettend goede moeders zullen zijn, en met hun natuurlijk Robbie en Joost, die fantastische vaders zullen zijn! Elsje, die er gelukkig altijd is.

Mijn begeleider Frans Kokkeler, voor onze eindeloze gesprekken over even zoveel onderwerpen. Ik ben ervan overtuigd dat jouw visie en stimulatie bepalend zijn geweest voor het resultaat van mijn promotie. Niet alleen voor de inhoud, maar ook zeker voor het plezier dat ik heb gehad tijdens de lange rit.

Snel daar achteraan komt Hans, die ook veel invloed heeft gehad op zowel de inhoud van mijn boekje als de weg er naartoe. De Paarse Bank, de gelijknamige wereld, je weet wel. Hopelijk kunnen we samen nog veel projecten doen! Georg Still, dank voor je wiskundige kijk op mijn onderzoek en discussies over "marriage"-problemen!

Mijn collega's van de vakgroep, die op gezette momenten een fijne discussie of gewoon slap geouwehoer mogelijk maakten. Bedankt voor de goede sfeer, die zich hopelijk uitstrekt in de toekomst! Een speciaal dankwoord voor de zingende Ajoo's: Gijs, Frederik en Irene. Onze muzikale ontdekkingstocht was leuk en verhelderend: zingen is niets voor mij.

Wessel, mijn walking-talking helpdesk en bovenal goede vriend. De enige die meteen weet *waar* in mijn LaTeX-code ik een uitroeptekentje, apostrofje dan wel minteken ben vergeten. Ongelofelijk. Enorm bedankt. Ontzettend leuk om samen te studeren, promoveren en hierna docenteren.

Rutger, ook jou dank ik voor je aanwezigheid tijdens de middelbare school, studie en promotie tijd. Op de volgende!

Juan, een fijne collega en vriend. Leuke discussies, let's keep it up! Maerten, the code-magician. MarcoO, el valorisator! Njels, de baas. Goed dat jullie er waren, zijn en zullen zijn! En de rest van de Ajoo's natuurlijk: Matthhijss, Jan, Denise-Denise, Boris und Der Krein. Maar ook in het Westen des Landsch, waar het sjaaltje van Hanz vrolijk wappert: Valentina.

De raggende stammetjes: Wout, Jens & Mijke, Richard en Thijz. Het Hooge Noorden, de Spiegel/Images. Ze roepen. Wout: bedankt voor je hulp met de voorkant! Looking good...

Natuurlijk mag de 67-clan niet vergeten worden: Nannoekie, Vincent, Germ, Jorien, Jop, Ronaldicus P., Casper Joost. De Groene Draeck-mannen: Sieger, Rutger en Robbie (zum zweiten Mal), Ronald en C! Vaker oud&nieuw diners en feesten!

Nou is het niet de bedoeling dat ik nog meer bladzijden volschrijf, dus ik moet er een eind aan breien. Ik heb genoten en hoop jullie allemaal snel weer te zien. Voor iedereen die ik niet met naam genoemd heb: ontzettend bedankt en tot snel!

List of References

[1] S.L. Ahire and P. Dreyfus. The impact of design management and process management on quality: an empirical investigation. *Journal of Operations Management*, 18: 549-575, 2000. [cited at p. 1]

[2] E.K. Antonsson and J. Cagan. *Formal Engineering Design Synthesis.* Cambridge University Press, 2001. [cited at p. 14, 17]

[3] Roman Barták. Constraint programming: in pursuit of the holy grail. *Proceedings of WDS99, Prague*, 1999. [cited at p. 15, 36, 39]

[4] J. Bento, B. Feijo, and D.L. Smith. Engineering design knowledge representation based on logic and objects. *Computers and Structures*, 63 - 5: 1015-1032, 1997. [cited at p. 31]

[5] F. Bolognini, A.A. Seshia, and K. Shea. Exploring the application of a mult-domain simulation-based computational synthesis method in mems design. *Proceedings of the 16th International Conference on Engineering Design, ICED'07*, 2007. [cited at p. 14]

[6] J. Cagan, M.I. Campbell, S. Finger, and T. Tomiyama. A framework for computational design synthesis: Models and applications. *Journal of Computing and Information Science in Engineering*, 5: 171-181, 2005. [cited at p. 6, 14]

[7] M.I. Campbell, J. Cagan, and K. Kotovsky. Agent-based synthesis of electromechanical design configurations. *Journal of Mechanical Design*, 122/61, 2000. [cited at p. 14]

[8] M.I. Campbell, J. Cagan, and K. Kotovsky. The a-design approach to managing automated design synthesis. *Research in Engineering Design*, 14: 12-24, 2003. [cited at p. 14]

[9] A. Chakrabarti. *Engineering Design Synthesis: Understanding, Approaches and Tools*, volume ISBN 1852334924. Springer Verlag, London, 2002. [cited at p. 14]

[10] H. Draijer and F.G.M. Kokkeler. Heron's synthesis engine applied to linkage design- the philosophy of watt software. *Proceedings of the Design Engineering Technical Conferences and Computer and Information in Engineering Conference, Montreal 2002*, DETC2002/MECH-34373, 2002. [cited at p. 19]

[11] N.F.O. Evbuomwan, S. Sivaloganathan, and A. Jebb. A survey of design philosophies, models, methods and systems. *Journal of Engineering Manufacture*, 210: 301-320, 1996. [cited at p. 26]

[12] E.A. Feigenbaum. How the "what" becomes the "how". *communications of the ACM*, 39, 1996. [cited at p. 17]

[13] E.A. Feigenbaum. Some challenges and grand challenges for computational intelligence. *Journal of the ACM*, 50, 2003. [cited at p. 17]

[14] D. Fensel. *The Knowledge Acquisition and Representation Language, KARL.* Kluwer Academic Publishers, 1995. [cited at p. 13, 17]

[15] W. Geerdes. Smart design of baggage handling systems. Master's thesis, University of Twente, Faculty of Engineering Design, OPM-816, 2007. [cited at p. 78]

[16] J.S. Gero and U. Kannengiesser. The situated function-behaviour-structure framework. *Design Studies*, 25(373-391), 2004. [cited at p. XV, 11, 12]

[17] R. Green. Cad manager survey 2004: Part 2. *Cadalyst (www.cadalyst.com)*, 2004. [cited at p. 1]

[18] R.M. Henderson and K.B. Clark. Architectural innovation: the reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 35(1990): 9-30, 1990. [cited at p. 1]

[19] V. Hubka and W.E. Eder. *Design Science.* Springer-Verlag, 1996. [cited at p. 9]

[20] S. Kirkpatrick, C.D. Gelatt jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 13: 671-980, 1983. [cited at p. 15]

[21] T.D. Kuczmarski. Managing new products: competing through excellence. *in: Rao199*, pp. 83-84, 1998. [cited at p. 1]

[22] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI magazine*, 13(1): 32-44, 1992. [cited at p. 15, 28]

[23] T. Kurtoglu and M.I. Campbell. A graph grammar based framework for automated concept generation. *Proceedings of the International Design Conference 2006*, pp. 61-68, 2006. [cited at p. 14, 105]

[24] R.T. Marler and J.S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26: 369-395, 2004. [cited at p. 15]

[25] W. Matek, D. Muhs, H. Wittel, and M. Becker. *Roloff-Matek machine-onderdelen*. Academic Service Schoonhoven, 1996. [cited at p. 20, 84]

[26] Ohsuga In: C. McMahon and J. Brown. *CAD/CAM principles, practise and manufacturing management*. Addison-Wesley Longman Ltd, 1998. [cited at p. XV, 26, 27]

[27] MOKA consortium Melody Stokes. *Managing Engineering Knowledge: MOKA*. Professional Engineering Publication, 2001. [cited at p. 16, 18]

[28] J. Miedema, M.C. van der Voort, and F.J.A.M. van Houten. Application of synthetic environments in product design. *CIRP Journal of Manufacturing Science and Technology*, 1: 159-164, 2009. [cited at p. 19]

[29] J. Moss, J. Cagan, and K. Kotovsky. Learning from design experience in an agent-based design system. *Research in Engineering Design*, 15: 77-92, 2004. [cited at p. 14]

[30] N. Nijenmanting. Towards a synthesis tool for baggage handling systems. Master's thesis, University of Twente, Faculty of Engineering Design, OPM-844, 2008. [cited at p. 78]

[31] P.Y. Papalambros and D.J. Wilde. *Principles of Optimal Design- Modeling and Computation*. Cambridge University press, 2000. [cited at p. 6]

[32] K.S. Pawar and J.C.K.H. Riedel. A survey of cad use in the uk mechanical engineering industry. *International Journal of Computer Applications in Technology*, 9(4): 219-228, 1996. [cited at p. 2]

[33] C.K. Prahalad and K. Lieberthal. The end of corporate imperialism. *Harvard Business Review*, 2003. [cited at p. 1]

[34] S.S. Rao, A. Nahm, S. Zhengzhong, X. Deng, and A. Syamil. *Artificial intelligence and expert systems applications in new product development- a survey*. Kluwer Academic Publishers, 1999. [cited at p. 1]

[35] B. Raphael and I.F.C. Smith. *Fundamentals of Computer-Aided Engineering*. Wiley, 2003. [cited at p. 17]

[36] R. Rothwell. Toward the fifth-generation innovation process. *International Marketing Review*, 11(1): 7-31, 1994. [cited at p. 1]

[37] O.W. Salomons. *Computer support in the Design of Mechanical Products.* Copyprint 2000, 1995. [cited at p. 18]

[38] K. Schilstra. *Towards continuous knowledge engineering. PhD Thesis.* 2003. [cited at p. 17]

[39] W.O. Schotborgh, F.G.M. Kokkeler, H. Tragter, M.J. Bommhoff, and F.J.A.M. van Houten. A generic synthesis algorithm for well-defined parametric design. *Proceedings of the 18th CIRP Design Conference*, 2008. [cited at p. 29]

[40] W.O. Schotborgh, F.G.M. Kokkeler, H. Tragter, and F.J.A.M. van Houten. A bottom-up approach for automated synthesis tools in the engineering design process. *Proceedings of International Design Conference 2006*, pp. 349-356, 2006. [cited at p. 19, 55]

[41] W.O. Schotborgh, H. Tragter, F.G.M. Kokkeler, F.J.A.M. van Houten, and T. Tomiyama. Towards a generic model of smart synthesis tools. *Proceedings of the CIRP Design Seminar 2007*, 2007. [cited at p. 19]

[42] T.W. Simpson, J.D. Peplinksi, P.N. Koch, and J.K. Allen. Metamodels for computer-based engineering design: survey and recommendations. *Engineering with Computers*, 2001. [cited at p. 2]

[43] P. Sridharan and M.I. Campbell. A grammar for function structures. *Proceedings of the ASME Design Engineering Technical Conference*, 3: 41-55, 2004. [cited at p. 14]

[44] A.C. Starling. Performance-based computational synthesis of parametric mechanical systems. *PhD thesis*, 2004. [cited at p. 14]

[45] S. Studer and V.R Benjamins. Knowledge engineering: principles and methods. *Data & Knowledge Engineering*, 25: 161-197, 1998. [cited at p. 18, 28]

[46] M. Tideman. *Scenario Based Product Design. PhD Thesis.* Printpartners Ipskamp, 2008. [cited at p. 19]

[47] T. Tomiyama and W.O. Schotborgh. Yet another model of design synthesis. *Proceedings of the 16th International Conference on Engineering Design, ICED'07*, pp. 83-84, 2007. [cited at p. 26, 106]

[48] H. Tragter, W.O. Schotborgh, M.H.L. Röring, and F.J.A.M. van Houten. Generic data architecture for parametric synthesis systems. *Proceedings of CIRP Design Conference 2008, Enschede*, 2008. [cited at p. 58, 106]

[49] D.G. Ullman. Toward the ideal mechanical engineering design support system. *Research in Engineering Design*, 13(2): 55-64, 2002. [cited at p. 2]

[50] Y. Umeda and T. Tomiyama. Fbs modeling: Modeling scheme of function for conceptual design. 1995. [cited at p. 10]

[51] C. Vempati and M.I. Campbell. A graph grammar approach to generate neural network topologies. *2007 Proceedings of the ASME Design Engineering Technical Conferences and Computers and Informatics in Engineering Conference*, 6: 79-89, 2008. [cited at p. 14]

[52] W. Visser. Designing as construction of representations: A dynamic viewpoint in cognitive design research. *Human-Computer Interaction*, 21: 103-152, 2006. [cited at p. 23, 35, 36, 96]

[53] B.J. Wielinga, A.T. Schreiber, and J.A. Breuker. Kads: a modelling approach to knowledge engineering. *Knowledge Acquisition*, 4: 5-53, 1992. [cited at p. 13]

[54] H. Yoshikawa. General design theory and a cad system. *Man-Machine Communication in CAD/CAM*, 1981. [cited at p. 10]

[55] M. Yoshioka, Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, and T. Tomiyama. Physical concept ontology for the knowledge intensive engineering framework. *Advanced Engineering Informatics*, 18: 95-113, 2004. [cited at p. 12]

[56] W.J. Zhang, Y. Lin, and N. Sinha. On the function-behavior-structure model for design. *Canadian Design Engineering Network conference 2005, CD*, 2005. [cited at p. XV, 10, 11]

Appendices

Appendix A

Synthesis knowledge

A.1 Optical chamber

This section states the knowledge base for synthesis, as implemented in the prototype for optical chamber design.

A.1.1 Elements and parameters

The elements of the optical chamber are stated in table A.1 and the parameters in table A.2. Figure A.1 illustrates the parameters concerning the optical axis.

Table A.1: *optical chamber, elements*

name	description
OC	optical chamber
T	tube
Det	detector
S	sample
C	casing
D	diaphragm
DSP	diaphragm set primary side
DSS	diaphragm set secondary side

Table A.2: *optical chamber, parameters*

type	name	description
scenario	S_{mat}	sample material
	T_A	tube surface area
	T_{mat}	tube material
	Det_A	detector surface area
embodiment	$T_{(x,y)}$	tube position
	T_α	tube orientation
	$Det_{(x,y)}$	detector position
	Det_α	detector orientation
	$S_{(x,y)}$	sample position
	S_d	sample diameter
	S_α	sample orientation
	S_t	sample thickness
	C_{mat}	casing material
	C_d	casing diameter
	C_L	casing length
	C_t	casing thickness
	D_{mat}	diaphragm material
	D_t	diaphragm thickness
	D_O	diaphragm aperture opening
	$D_{(x,y)}$	diaphragm aperture center
	D_α	diaphragm orientation
	D_k	diaphragm knife edge
auxiliary	$OC_{\beta P}$	angle primary axis
	OC_{LP}	length primary axis
	$OC_{\beta S}$	angle secondary axis
	OC_{LP}	length secondary axis
	$OC_{(x,y)}$	optical center (x, y)
	OC_{Emax}	maximum energy
	OC_{Emin}	minimum energy
	OC_{NT}	total number of diaphragms
	OC_{NP}	number of primary diaphragms
	OC_{NS}	number of secondary diaphragms
	$OC_{midpoint}$	focus on sample midpoint (y/n)
	S_{VA}	sample view area
	$S_{standard}$	sample standard (y/n)
	DSP_{sample}	DSP against sample (y/n)
	DSP_{tube}	DSP against tube (y/n)
	DSS_{sample}	DSS against sample (y/n)

Continued on next page

Table A.2 – continued from previous page

type	name	description
	$DSS_{detector}$	DSS against detector (y/n)

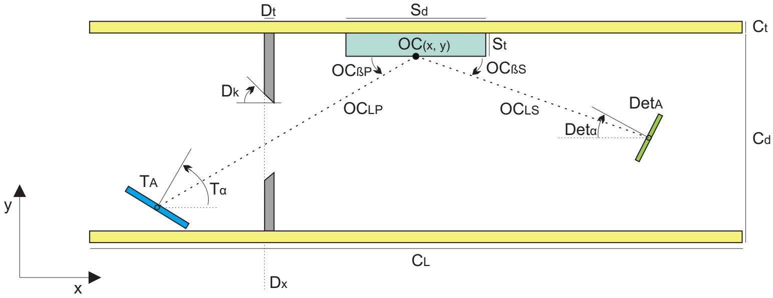


Figure A.1: *optical axis*

A.1.2 X-rules

The X-rules are stated in set A.1. The required information is included on the left side of the arrow, and any parameter values that are resolved are indicated on the right hand side.

$$
\begin{aligned}
& OC && \xrightarrow{expands} T, Det, S, C, DSP, DSS \qquad (A.1) \\
& DSP(DSP_{sample}, DSP_{tube}, OC_{NP}, \\
& \quad S_{(x)}, S_d, T_{(x)}, T_\alpha, T_A) && \xrightarrow{expands} D(x) \\
& DSS(DSS_{sample}, DSS_{detector}, OC_{NS}, \\
& \quad S_{(x)}, S_d, Det_{(x)}, Det_\alpha, Det_A) && \xrightarrow{expands} D(x)
\end{aligned}
$$

A.1.3 R-rules

The R-rules are stated in equation set (A.2).

$$
\begin{aligned}
OC_{(x,y)} &= f(OC_{midpoint}, S_{(x,y)}, S_t) && \rightarrow OC_{(x,y)} \qquad (A.2) \\
OC_{NT} &= OC_{NP} + OC_{NS} && \rightarrow OC_{NT}, OC_{NP}, OC_{NS} \\
OC_{Emax} &= f(T_{mat}) && \rightarrow OC_{Emax}
\end{aligned}
$$

$$
\begin{aligned}
OC_{Emin} &= f(T_{mat}) && \rightarrow OC_{Emin} \\
T_{(x)} &= OC_{(x)} - OC_{LP} \cdot \cos(OC_{\beta P}) && \rightarrow T_{(x)} \\
T_{(y)} &= OC_{(y)} - OC_{LP} \cdot \sin(OC_{\beta P}) && \rightarrow T_{(y)} \\
T_{\alpha} &= f(OC_{\beta P}) && \rightarrow T_{\alpha} \\
Det_{(x)} &= OC_{(x)} + OC_{LS} \cdot \cos(OC_{\beta S}) && \rightarrow Det_{(x)} \\
Det_{(y)} &= OC_{(y)} - OC_{LS} \cdot \sin(OC_{\beta S}) && \rightarrow Det_{(y)} \\
Det_{\alpha} &= f(OC_{\beta S}) && \rightarrow Det_{\alpha} \\
S_{(x,y)} &= f(S_t, OC_{(x,y)}, OC_{midpoint}) && \rightarrow S_{(x,y)} \\
C_d &= f(T_{(y)}, T_A, T_{\alpha}, Det_{(y)}, Det_A, Det_{\alpha}, S_{(y)}, S_t) && \rightarrow C_d \\
C_L &= f(T_{(x)}, T_A, T_{\alpha}, Det_{(x)}, Det_A, Det_{\alpha}) && \rightarrow C_L \\
C_t &= f(C_{Emax}, C_{mat}) && \rightarrow C_t \\
(D_O + D_{(y)}) &= f(D_{(x)}, D_t, D_k, OC_{(x,y)}, S_d, S_{VA}, \\
&\qquad T_{(x,y)}, T_A, T_{\alpha}, Det_{(x,y)}, Det_A, Det_{\alpha}) && \xrightarrow{1} (D_O + D_{(y)}) \\
D_t &= \frac{\ln(0.1)}{-T_{mat.\mu} \cdot T_{mat.\rho}} && \xrightarrow{2} D_t \\
D_k &= f(OC_{\beta S}, OC_{\beta P}) && \rightarrow D_k
\end{aligned}
$$

notes

1. parameters $D_O + D_{(y)}$ are resolved simultaneously;
2. $mat.\mu$ is the attenuation coefficient of a material, $mat.\rho$ is the density.

A.1.4 C-rules

C-rules are given in equation set (A.3).

$$
\begin{aligned}
\sin(OC_{\beta P}) &\geq \frac{\frac{T_A}{2} \cdot \sin(T_{\alpha})}{OC_{LP}} && \rightarrow OC_{\beta P}, OC_{LP}, T_A, T_{\alpha} \qquad \text{(A.3)} \\
\cos(OC_{\beta P}) &\leq \frac{\frac{S_d}{2} + OC_{NP} \cdot D_t + \frac{T_A}{2 \cdot \cos(T_{\alpha})}}{OC_{LP}} && \rightarrow OC_{\beta P}, OC_{LP}, OC_{NP}, T_A, T_{\alpha}, S_d \\
\sin(OC_{\beta S}) &\geq \frac{\frac{Det_A}{2} \cdot \sin(Det_{\alpha})}{OC_{LS}} && \rightarrow OC_{\beta S}, OC_{LS}, Det_A, Det_{\alpha} \\
\cos(OC_{\beta S}) &\leq \frac{\frac{S_d}{2} + OC_{NS} \cdot D_t + \frac{Det_A}{2 \cdot \cos(Det_{\alpha})}}{OC_{LS}} && \rightarrow OC_{\beta S}, OC_{LS}, OC_{NS}, Det_A, Det_{\alpha}, S_d
\end{aligned}
$$

A.2 Baggage handling systems

This section presents the synthesis knowledge as implemented in the prototype for baggage handling system (BHS) design.

The discussion on elements, parameters, X-rules and R-rules is divided into five categories, table A.3.

Table A.3: *categories*

category	**description**
PFD process	main processes of a BHS
EquipmentGroupSet	contains multiple equipment groups
EquipmentGroup	contains multiple pieces of equipment
Equipment	handles the baggage
Miscellaneous	elements Area, BeltNetwork and BaggageFlow

A.2.1 Elements

In total there are 51 elements, as summarized in table A.4 and elaborated further in this section.

Table A.4: *elements*

name	**quantity**
PFD process	10
EquipmentGroupSet	9
EquipmentGroup	9
Equipment	19
Miscellaneous	4

PFD process The BHS functional design is described with the 10 PFD process elements stated in table A.5.

Table A.5: *PFD process elements*

name	description
CheckIn	check-in belt (BHS input)
HBS12	hold baggage screening level 1 + 2
HBS34	hold baggage screening level 3 + 4
HBS5	hold baggage screening level 5
SuspectBagSplit	split baggage flow into cleared and suspect
BagRemoval	removes bag from system (BHS output)
DetermineID	determine the ID tag on baggage
IDReadSplit	split baggage flow into ID read and not read
ManualCoding	manual determination of ID tag on baggage
Sortation	sort baggage to chutes (BHS output)

Equipment/Group/Set Each PFD process element contains an element XXEquipmentGroupSet, where XX is the name of the PFD process. Within this set, there are multiple elements XXEquipmentGroup. The only exception is CheckIn, which does not have EquipmentGroupSet or EquipmentGroup. For the others, this introduces 19 elements.

The EquipmentGroup element contains the actual equipment that operate on the baggage flow. The list of 19 equipment elements is given in table A.6.

Table A.6: *equipment elements*

PFD process	name	description
general	Belt	contains baggage flow
	FeedInBelt	feed baggage into equipment
	Divert12	split one flow into two
	Merge21	merge two flow into one
HBS12	ScreeningMachineLevel1	level 1 screening
	ScreeningMachineLevel2	level 2 screening
HBS34	ScreeningMachineLevel3	level 3 screening
	ScreeningMachineLevel4	level 4 screening
HBS5	ScreeningMachineLevel5	level 5 screening
SuspectBagSplit	SuspectBagDivertInput	split flow into suspect and cleared
	SuspectBagDivertExitSuspect	divert exit, suspect
	SuspectBagDivertExitCleared	divert exit, cleared

Continued on next page

Table A.6 – continued from previous page

PFD process	name	description
BagRemoval	BagRemovalEquipment	removes baggage from BHS
DetermineID	DetermineIDMachine	reads ID tags on baggage
IDReadSplit	IDReadDivertInput	splits flow into ID read and ID not read
	IDReadDivertExitRead	divert exit, ID read
	IDReadDivertExitNotRead	divert exit, ID not read
ManualCoding	ManualCodingStation	reads ID tags manually
Sortation	Sorter	sorts baggage to exit chutes

Miscellaneous Contains the 4 elements of table A.7.

Table A.7: *miscellaneous elements*

name	description
Area	the "mother-element" that contains the PFD processes
BeltNetwork	collects incoming baggage flow for PFD process
BaggageFlow	contains the belts
Network	redistributes the baggage flow over belts

A.2.2 Parameters

BHS synthesis knowledge contains 204 parameters, summarized in table A.8 and elaborated in the following sections.

Table A.8: *parameters*

name	quantity
PFD process	55
EquipmentGroupSet	9
EquipmentGroup	9
Equipment	128
Miscellaneous	3

PFD process Table A.9 lists the 6 “generic” parameters for the PFD processes. Exceptions are listed in table A.10. The total amount of parameters on PFD process level: (10 PFD processes)·(6 parameters) - 5(see Table A.10) = 55.

Table A.9: *PFD process parameters*

name	description
totalFlow	total baggage flow through PFD process
maxFlow	maximum baggage flow
numberOfGroups	number of equipment groups
numberOfInputBelts	number of input belts
numberOfOutputBelts	number of output belts
connectedToID	connection data

Table A.10: *modifications*

name	modification
CheckIn	minus numberOfGroups, numberOfInputBelts, numberOfOutputBelts, connectedToID
Sortation	minus numberOfOutputBelts
BagRemoval	minus numberOfOutputBelts
DetermineID	added onlyClearedBaggage y/n

Equipment/Group/Set The EquipmentGroupSet and EquipmentGroup all have a single parameter: connectedToID. Resulting in 18 parameters.

The equipment elements have parameters, as specified in table A.11 (ScreeningMachineLevelN is abbreviated to ScreeningN and SuspectBagDivert into SuspectBD).

Table A.11: *equipment parameters*

equipment	name	description
Belt	connectedToID totalFlow	network connection total baggage flow

Continued on next page

Table A.11 – continued from previous page

equipment	name	description
	maxFlow	maximum baggage flow
	suspect	amount of suspect baggage
	cleared	amount of cleared baggage
	IDRead	amount of ID read baggage
	IDNotRead	amount of ID not read baggage
FeedInBelt	*idem Belt*	*idem Belt*
Divert12	connectedToID	network connection
	ratio	ratio left/right baggage flow
Merge21	connectedToID1	network connection
	connectedToID2	network connection
Screening1	connectedToID	network connection
	totalFlow	total baggage flow
	maxFlow	maximum baggage flow
	suspect	amount of suspect baggage
	cleared	amount of cleared baggage
	IDRead	amount of ID read baggage
	IDNotRead	amount of ID not read baggage
	percentCleared	percentage of baggage cleared
Screening2	*idem Screening1*	*idem Screening1*
Screening3	*idem Screening1*	*idem Screening1*
Screening4	*idem Screening1*	*idem Screening1*
Screening5	*idem Screening1*	*idem Screening1*
SuspectBDInput	*idem Belt*	*idem Belt*
SuspectBDExitSuspect	*idem Belt*	*idem Belt*
SuspectBDExitCleared	*idem Belt*	*idem Belt*
BagRemovalEquipment	connectedToID	network connection
	totalFlow	total baggage flow
	maxFlow	maximum baggage flow
DetermineIDMachine	*idem Belt*	*idem Belt*
	percentageRead	percentage of ID tags read
IDReadDivertInput	*idem Belt*	*idem Belt*
IDReadDivertExitRead	*idem Belt*	*idem Belt*
IDReadDivertExitNotRead	*idem Belt*	*idem Belt*
ManualCodingStation	*idem Belt*	*idem Belt*
	percentageRead	percentage of ID tags read
Sorter	*idem Belt*	*idem Belt*
	numberOfExits	number of exit chutes
	length	length of the sorter

The equipment parameters: (11 · 7 Belt parameters) + (5 · 8 Screening1 parameters) + 11 = 128 parameters.

Miscellaneous Only the element BaggageFlow has three parameters: numberOfBelts, connectedToID and totalFlow.

A.2.3 X-rules

A total of 46 X-rules are used to construct the topological tree: table A.12, and discussed in categories. The X-rules also resolve the connectedToID parameter for their sub-elements.

Table A.12: *X-rules*

category	**quantity**
PFD process	27
EquipmentGroupSet	9
EquipmentGroup	9
Equipment	0
Miscellaneous	1

PFD process PFD processes have, in general, three X-rules:

1. expand and connect BeltNetwork, EquipmentGroupSet and BaggageFlow;

2. expand and connect Belts in BeltNework (to collect baggage flow from upstream PFD processes);

3. expand and connect Belts in BaggageFlow (the output side).

Exceptions are CheckIn, which does not have the second type, and BagRemoval and Sortation, which do not have the third type. In total, that brings 27 X-rules on PFD process level.

Equipment/Group/Set All EquipmentGroupSet elements have one R-rule to expand and connect with the required number of EquipmentGroups. In turn, these elements expand and connect their specific configuration of equipment. For these elements there are in total 18 X-rules.

Miscellaneous BeltNetwork expands its topology with two BaggageFlows and a Network in between: one X-rules.

A.2.4 R-rules

120 R-rules are required for synthesis of BHS design: table A.13 and discussed in categories.

Table A.13: *R-rules*

category	**quantity**
PFD process	35
EquipmentGroupSet	0
EquipmentGroup	0
Equipment	83
Miscellaneous	2

PFD process The "generic" R-rules for PFD processes are four in total, table A.14.

Table A.14: *parameters and R-rules*

parameter	**R-rule**
totalFlow	sum the flows inside BaggageFlow of BeltNetwork
numberOfGroups	divide totalFlow with maxFlow of equipments
numberOfInputBelts	count number of input belts in first equipment of each EquipmentGroup
numberOfOutputBelts	count number of output belts in last equipment of each EquipmentGroup

Exceptions are CheckIn, which has no R-rules, and BagRemoval which has no output belts. This results in 10 · 4 - 5 = 35 R-rules.

Equipment/Group/Set The elements EquipmentGroupSet and EquipmentGroup have no R-rules, but equipment does. The five "generic" R-rules for equipment concern the baggage flow. Each flow (suspect, cleared, ID read and ID not read) is retrieved from the upstream equipment and processed, if necessary (e.g. suspect becomes cleared for a screening machine). The parameter totalFlow is a summation of the suspect and cleared baggage flow.

Exceptions are:

1. BaggageRemovalEquipment: only one R-rule for totalFlow: resolved by retrieving from upstream equipment;

2. Divert12: no R-rules;
3. Merge21: no R-rules;
4. Sorter: in addition resolves sorterLength and numberOfExits from totalFlow.

This brings the number of equipment R-rules to: $19 \cdot 5$ - $12 = 83$.

Miscellaneous BaggageFlow has two R-rules: for numberOfBelts and totalFlow.

A.3 Compression spring

This section states the knowledge base for synthesis, as implemented in the prototype for compression springs.

A.3.1 Parameters

The parameters are given in table A.15.

Table A.15: *compression spring, parameters*

type	**name**	**description**
performance	F	force at compression
	W	stored energy at compression
scenario	s	compression
	load type	load type
embodiment	D	mean diameter of spring body
	L_0	length of spring body
	n_{total}	total coils
	mat	material
	d	wire diameter (DIN specified)
	end finish	end finish
auxiliary	L_s	length at compression
	R	spring constant
	s_{max}	maximum compression
	L_{min}	minimum operational length
	n_{active}	active coils
	w	winding ratio
	S_a	minimum space between coils
	D_i	internal diameter

Continued on next page

Table A.15 – continued from previous page

type	name	description
	D_e	external diameter
	L_c	bloc length
	s_c	bloc compression
	F_{max}	force at bloc compression
	σ	stress at bloc compression
	σ_{max}	maximum allowed stress

A.3.2 R-rules

R-rules are given in equation set (A.4).

$$
\begin{aligned}
R &= \frac{mat.G}{8} \cdot \frac{d^4}{D^3 \cdot n_{active}} && \rightarrow R, D, n_{active} \\
R &= \frac{F}{s} && \rightarrow R, F, s \\
W &= \frac{R \cdot s^2}{2} && \rightarrow W, R, s \\
\sigma_{max} &= 0.45 \cdot (mat.Rm1 - mat.Rm2 \cdot \log(d)) && \rightarrow \sigma_{max} \\
L_{min} &= L_c + S_a && \rightarrow L_{min}, L_c, S_a \\
L_0 &= s_c + L_c && \rightarrow L_0, s_c, L_c \\
d &= k_1 \cdot \sqrt[3]{F \cdot D_e} && \xrightarrow{1} d \\
n_{total} &= n_{active} + \text{factor} && \xrightarrow{2} n_{total}, n_{active} \\
S_a &= (0.0015 \cdot \frac{D^2}{d} + 0.1 \cdot d) \cdot n_{active} && \\
&= 0.02 \cdot (D + d) \cdot n_{active} && \\
&= 1.5 \cdot S_a && \\
&= 2 \cdot S_a && \xrightarrow{3} S_a, D, n_{active} \\
w &= \frac{D}{d} && \rightarrow w, D \\
D &= \frac{D_i + D_e}{2} && \rightarrow D, D_i, D_e \\
D_i &= D - d && \rightarrow D_i, D \\
D_e &= D + d && \rightarrow D_e, D \\
F_{max} &= R \cdot s_c && \rightarrow F_{max}, R, s_c
\end{aligned}
\tag{A.4}
$$

$$\begin{aligned}
\sigma &= F_{max}\cdot \frac{D}{0.4\cdot d^3} && \rightarrow \sigma, F_{max}, D \\
L_c &= (n_{total} + \text{factor})\cdot d && \xrightarrow{4} L_c, n_{total} \\
L_0 &= s + L_s && \rightarrow L_0, s, L_s \\
L_0 &= s_{max} + L_{min} && \rightarrow L_0, s_{max}, L_{min}
\end{aligned}$$

notes

1. k_1 depends on material, and the calculated wire diameter is matched to the nearest in the DIN specification;

2. factor depends on material, either 2 or 1.5;

3. these rules are considered together, taking into account the load type and material. The resolved parameters are S_a, D or $n_a ctive$;

4. factor depends on material and end finish.

A.3.3 C-rules

C-rules are given equation set (A.5).

$$\begin{aligned}
d &\geq mat.d_{min} && \rightarrow d, mat \\
d &\leq mat.d_{max} && \rightarrow d, mat \\
D &\leq mat.D_{max} && \rightarrow D, mat \\
L_0 &\leq mat.L_{max} && \rightarrow L_0, mat \\
n_{active} &\geq mat.n_{min} && \rightarrow n_{active}, mat \\
w &\geq mat.w_{min} && \rightarrow w, mat \\
w &\leq mat.w_{max} && \rightarrow w, mat \\
\sigma &\leq \sigma_{max} && \rightarrow \sigma, \sigma_{max} \\
F &\leq F_{max} && \rightarrow F, F_{max} \\
L_c &\leq L_{min} && \rightarrow L_c, L_{min} \\
L_{min} &\leq L_0 && \rightarrow L_{min}, L_0 \\
s &\leq s_{max} && \rightarrow s, s_{max} \\
L_0 &\geq d\cdot n_{total} && \rightarrow L_0, d, n_{total} \\
L_s &\geq L_{min} && \rightarrow L_s, L_{min} \\
s_{max} &\leq s_c && \rightarrow s_{max}, s_c
\end{aligned} \tag{A.5}$$

A.4 Extension spring

This section states the knowledge base for synthesis, as implemented in the prototype for extension springs.

A.4.1 Parameters

Table A.16: *extension spring, parameters*

type	name	description
performance	F	force at extension
	W	stored energy at extension
scenario	s	extension
	load type	load type
embodiment	D	mean diameter of spring body
	L_0	total length
	n_{total}	total coils
	n_{active}	active coils
	mat	material
	d	wire diameter (DIN specified)
	end finish	end finish type
	F_0	pre-load
auxiliary	R	spring constant
	L_k	length of active spring body
	D_i	internal diameter
	D_e	external diameter
	L_h	length of end-finishes
	L	length at extension
	s_{max}	maximum extension
	L_{max}	length at maximum extension
	F_{max}	force at maximum extension
	σ	stress at maximum extension
	σ_{max}	maximum allowed stress
	w	winding ratio

A.4.2 R-rules

R-rules are given in equation set (A.6).

$$\begin{aligned}
R &= \frac{mat.G}{8}\cdot \frac{d^4}{D^3\cdot n_{active}} && \rightarrow R, D, n_{active} \\
R &= \frac{F - F_0}{s} && \rightarrow F, F_0, R, s \\
W &= \frac{R\cdot s^2}{2} && \rightarrow W, R, s \\
\sigma &= F_{max}\cdot \frac{D}{0.4\cdot d^3} && \rightarrow \sigma, D, F_{max} \\
\sigma_{max} &= 0.45\cdot (mat.Rm1 - mat.Rm2\cdot \log(d)) && \rightarrow \sigma_{max} \\
L_{max} &= L_0 + s_{max} && \rightarrow L_{max}, L_0, s_{max} \\
L &= L_0 + s && \rightarrow L, L_0, s \\
L_h &= \text{factor}\cdot D_i && \xrightarrow{1} L_h \\
d &= k_1\cdot \sqrt[3]{F\cdot D_e} && \xrightarrow{2} d \\
n_{total} &= n_{active} + \text{factor} && \xrightarrow{3} n_{total}, n_{active} \\
w &= \frac{D}{d} && \rightarrow w, D \\
D &= \frac{D_i + D_e}{2} && \rightarrow D, D_i, D_e \\
D_i &= D - d && \rightarrow D_i, D \\
D_e &= D + d && \rightarrow D_e, D \\
F_{max} &= R\cdot s_{max} + F_0 && \rightarrow F_{max}, R, s_{max}, F_0 \\
L_k &= n_{active}\cdot d && \rightarrow L_k, n_{active} \\
L_0 &= L_k + 2\cdot L_h && \rightarrow L_0, L_k, L_h
\end{aligned} \tag{A.6}$$

notes

1. factor depends on end finish type and wire diameter;
2. k_1 depends on material and the calculated wire diameter is matched to the nearest in the table;
3. factor depends on end finish.

A.4.3 C-rules

C-rules are given in equation set (A.7).

$$
\begin{aligned}
d &\geq mat.d_{min} && \rightarrow d, mat \\
d &\leq mat.d_{max} && \rightarrow d, mat \\
D &\leq mat.D_{max} && \rightarrow D, mat \\
L_0 &\leq mat.L_{max} && \rightarrow L_0, mat \\
n_{active} &\geq mat.n_{min} && \rightarrow n_{active}, mat \\
w &\geq mat.w_{min} && \rightarrow w, mat \\
w &\leq mat.w_{max} && \rightarrow w, mat \\
F_0 &\leq \frac{0.075 - 0.00375 \cdot w}{0.45} \cdot \sigma_{max} \cdot \frac{0.4 \cdot d^3}{D} && \rightarrow F_0 \\
\sigma &\leq \sigma_{max} && \rightarrow \sigma, \sigma_{max} \\
F &\leq F_{max} && \rightarrow F, F_{max} \\
L_k &\leq L_0 && \rightarrow L_k, L_0 \\
L_0 &\leq L && \rightarrow L_0, L \\
s &\leq s_{max} && \rightarrow s, s_{max} \\
L_0 &\geq 2 \cdot D && \rightarrow L_0, D
\end{aligned}
\tag{A.7}
$$

A.5 Torsion spring

This section states the knowledge base for synthesis, as implemented in the prototype for torsion springs.

A.5.1 Parameters

Table A.17: *torsion spring, parameters*

type	name	description
performance	M	torque at rotation angle
	F	force at rotation angle
	$\sigma_{rotation}$	stress at rotation angle
	safety factor	safety factor on stress at rotation angle
scenario	α	rotation angle
embodiment	D	mean diameter of spring body
	L_0	length of spring body

Continued on next page

Table A.17 – continued from previous page

type	name	description
	n	number of coils
	mat	material
	d	wire diameter (DIN specified)
	L_{leg}	leg length
auxiliary	w	winding ratio
	D_i	internal diameter
	D_e	external diameter
	q	stress peaking factor
	L_{wire}	length of (unwound) wire
	a	distance between coils
	σ_{max}	maximum allowed stress

A.5.2 R-rules

R-rules are given in equation set (A.8).

$$
\begin{aligned}
L_{wire} &= D\cdot \pi\cdot n && \\
&= n\cdot \sqrt{(D\cdot \pi)^2 + (a+d)^2} && \xrightarrow{1} L_{wire}, D, n, a \\
L_0 &= (n+1.5)\cdot d && \\
&= n\cdot (a+d) + d && \xrightarrow{2} L_0, n, a \\
q &= \frac{w+0.07}{w-0.75} && \rightarrow q, w \\
M &= F\cdot L_{leg} && \rightarrow M, F, L_{leg} \\
\sigma_{rotation} &= M\cdot \frac{q}{\frac{\pi}{32}\cdot d^3} && \rightarrow \sigma_{rotation}, M, q \\
\alpha &= \frac{M\cdot L_{wire}}{mat.E\cdot \frac{\pi}{64}\cdot d^4} && \rightarrow \alpha, L_{wire}, M \\
\text{safety factor} &= \frac{\sigma_{rotation}}{\sigma_{max}} && \rightarrow \text{safety factor}, \sigma_{rotation} \\
w &= \frac{D}{d} && \rightarrow w, D \\
D &= \frac{D_i + D_e}{2} && \rightarrow D, D_i, D_e \\
D_i &= D - d && \rightarrow D_i, D
\end{aligned}
\tag{A.8}
$$

$$
\begin{aligned}
D_e &= D + d && \rightarrow D_e, D \\
d &= k_1 \cdot \frac{\sqrt[3]{F \cdot L_{leg}}}{1 - k_2} && \\
k_2 &= 0.06 \cdot \frac{\sqrt[3]{M}}{D_i} && \xrightarrow{3} d \\
\sigma_{max} &= 0.7 \cdot (mat.Rm1 - mat.Rm2) \cdot \log(d) && \rightarrow \sigma_{max} \\
a &= (0.24 \cdot w - 0.64) \cdot d^{0.83} && \rightarrow a, w
\end{aligned}
$$

notes

1. if $(a + d) \leq \frac{D}{4}$: use first equation, else use second;
2. if $a = 0$: use first equation, else use second;
3. if $d \leq 5mm : k_1 = 0.22$, else $k_1 = 0.24$;

A.5.3 C-rules

C-rules are given in equation set (A.9)

$$
\begin{aligned}
d &\geq mat.d_{min} && \rightarrow d, mat \\
d &\leq mat.d_{max} && \rightarrow d, mat \\
D &\leq mat.D_{max} && \rightarrow D, mat \\
L_0 &\leq mat.L_{max} && \rightarrow L_0, ma \\
L_0 &\geq n \cdot d && \rightarrow L_0, n, d \\
n &\geq mat.n_{min} && \rightarrow n, mat \\
w &\geq mat.w_{min} && \rightarrow w, mat \\
w &\leq mat.w_{max} && \rightarrow w, mat \\
\sigma_{rotation} &\leq \sigma_{max} && \rightarrow \sigma, \sigma_{max} \\
L_{leg} &\leq 5 \cdot D && \rightarrow L_{leg}, D
\end{aligned}
\tag{A.9}
$$