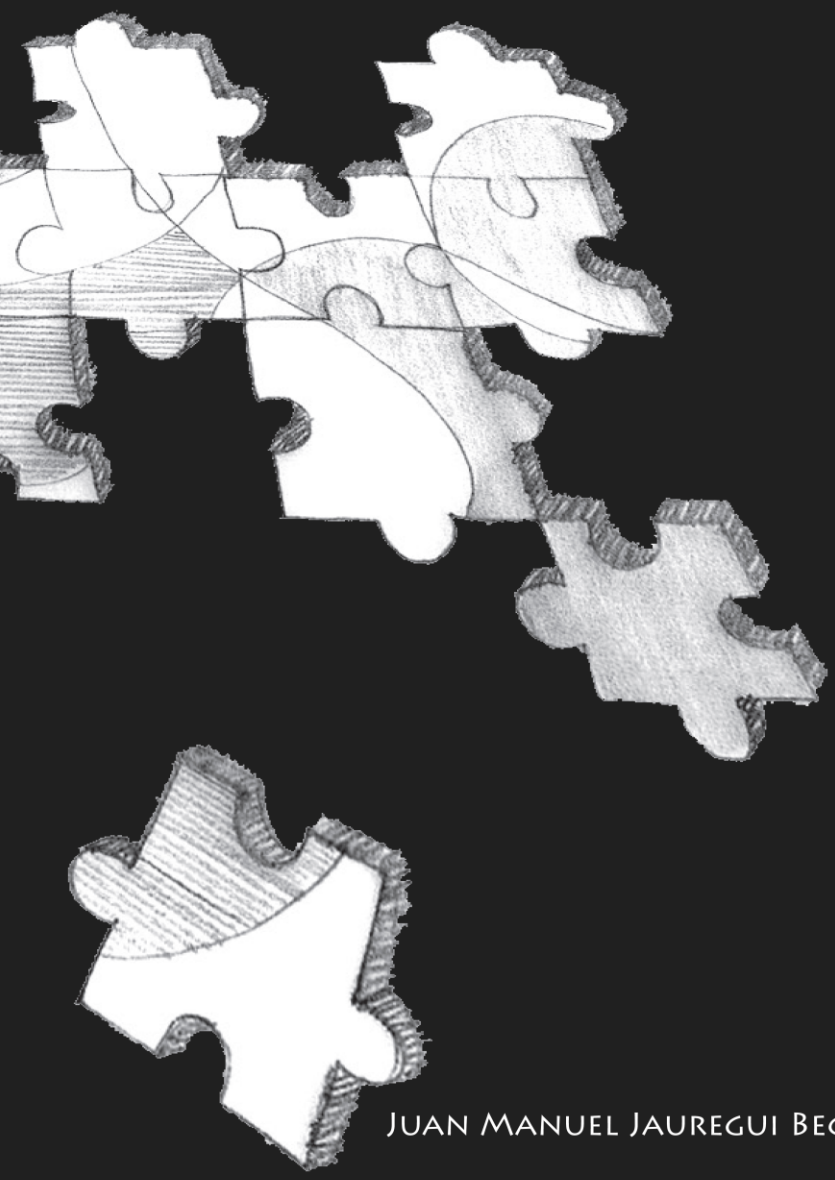
FROM HOW MUCH TO HOW MANY

MANAGING COMPLEXITY IN ROUTINE DESIGN AUTOMATION



JUAN MANUEL JAUREGUI BECKER

FROM HOW MUCH TO HOW MANY
Managing Complexity in Routine 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 donderdag 29 April 2010 om 16.45 uur

door

Juan Manuel Jauregui Becker
geboren op 20 May 1980
te Merida, Venezuela

Dit proefschrift is goedgekeurd door:

Prof. dr. ir. F.J.A.M. van Houten promotor

FROM HOW MUCH TO HOW MANY
Managing Complexity in Routine Design Automation

PhD Thesis

By Juan Manuel Jauregui Becker at the Faculty of Engineering Technology (CTW) of the University of Twente, Enschede, the Netherlands.

Enschede, 29 April 2010

De promotiecommissie:

Prof. dr. F. Eising	Universiteit Twente, voorzitter, secretaris
Prof. dr. ir. F.J.A.M. van Houten	Universiteit Twente, promotor
Dr. ir. G. Still	Universiteit Twente
Prof. dr. ir. R. Akkerman	Universiteit Twente
Prof. dr. ir. J. van Hillegersberg	Universiteit Twente
Prof. dr. ir. F.W. Jansen	Technische Universiteit Delft
Prof. dr. T. Tomiyama	Technische Universiteit Delft
Prof. dr. ir. A.C. Brombacher	Technische Universiteit Eindhoven
Prof. dr. T. Kjellberg	KTH Royal Institute of Technology

Keywords: Computational Design Synthesis, Routine Design, Complexity Management

ISBN 978-90-365-2989-1

A Diru,
a mis papas y hermanos

Summary

Advances in technology and competitive markets are driving the development of products with shorter times to market. In addition to this, products are becoming more complex, with more functionality, and yet lower prices. This has motivated the development of computer systems that support different phases of the design process, one of which is the synthesis process. This process consists of generating candidate design solutions to design problems.

This thesis researches the development of software that supports synthesis processes. This type of software is denoted in this thesis as Computer Aided Synthesis, or CAS. The scope lies within the boundaries of routine design problems of artifacts. In such problems, designers have all knowledge available about the components, parameters, relations and constrains that are used for solving them.

Synthesis processes for generating solutions to routine design problems consist of two tasks: (1) generating networks of components and (2) attributing values to unknown parameters. The exact strategy determining how these tasks are performed depends on the distribution of requirements throughout the different levels of detail of the problem, e.g. only on top, only at the bottom or as a mix. However, this relation is not known a priori and is different for different distributions of design requirements. From here that complexity in routine design is attributed to the distribution of design requirements along the different abstractions of the problem. Determining the dependency between design complexity and synthesis strategies is the main challenge of this research. The result is a new complexity management approach: *Computational Design Synthesis by Complexity Management*. This approach integrates six methods, which have all been developed during this research.

The first step of this approach consists in formulating a design problem using the *FBS based Formulation* method. This method uses FBS representations to aid designers in the process of determining which are the exact components, parameters, relations and constraints playing a role in the design problem.

The formal design problem is then decomposed by applying the *ADT based*

Decomposition method. This results in a new problem formulation consisting different levels of abstraction. The decomposition is both functional and physical and is based in the structure of the functions, components, parameters and relations in the problem.

The decomposed model is then translated into a *Topology Abstraction Representation Diagram* (TARD). TARD consists of four building blocks: Elements, C-relations, H-relations and ACO-relations. Elements represent individual components of the design problem, and group the set of parameters used in its description. C-relations represent the connectedness of the elements in the topology. H-relations model how a group of C-relations are related to describe the composition of a higher level element. ACO-relations are used to model analysis relations, physical coherence constraints and objective functions. The two most important characteristics of TARD are: it supports the representation of different distributions of design requirements for one problem, and it supports top-down and bottom-up synthesis strategies.

The next step in the approach is to determine the *Topology System of Equations* (ToSE) of the TARD model. ToSE consists of algebraic equations that define how the instantiation of one component is constrained by the instantiation of other components. These equations model the relation between components within one level of detail (balance equations), as well as between different levels of detail (vertical equations) in one TARD model. This method was developed in this research with the goal of translating topology characteristics of design problems into equations that can be handled by existing constraint solving algorithms.

Determining a synthesis strategy is done in this approach by applying the *Local Grammar Method* and the *KGM Solving Algorithm*. The first method specifies how to generate networks of components in one level of detail. The second algorithm determines how to proceed within different levels of abstraction by solving ToSE. Furthermore, this algorithm also determines the order in which the design parameters are solved.

Several examples describe the application of these techniques, while two software implementations demonstrate how the collaborative usage of these techniques leads to the automation of a routine design problem. The first implementation automates the design of cooling systems for injection molding. By combining these methods with specialized algorithms, the software is capable of automatically generating cooling systems for a given mold. A user interface developed with SolidWorks© API allows users to input their problems by specifying the geometry of the mold and its characteristics. The second implementation is a toolbox that implements *TARD*, *ToSE*, the *Local Grammar Method* and the *KGM Solving Algorithm* in a generic fashion. After entering the TARD model of a given design problem, this toolbox is capable of automatically generating candidate solutions.

The developed approach has the advantage that it permits the reuse of existing problem formulations, the use of standard solving methods, and the development of computer based design tools.

Samenvatting

Technologische ontwikkelingen en economische concurrentie zijn de aanjagers van een verkorting van de time to market. Daarnaast krijgen producten meer functionaliteit, stijgt hun complexiteit en staan prijzen onder druk. Deze tendensen zijn de drijfveren voor de ontwikkeling van computersystemen die verschillende fasen van het ontwerpproces ondersteunen. Een van die fasen is het synthese proces, waarin kandidaat-oplossingen voor ontwerpproblemen worden ontworpen.

Dit proefschrift onderzoekt de ontwikkeling van software die synthese processen ondersteunt. Dit type software wordt in dit proefschrift aangeduid als Computer Aided Synthesis, of CAS. Het toepassingsgebied ligt binnen de grenzen van het routinematige ontwerpen van artefacten. In dergelijke problemen hebben ontwerpers alle kennis beschikbaar over te gebruiken componenten, parameters, relaties en de beperkingen die gelden voor het toepassen.

Synthese processen voor het genereren van oplossingen voor problemen in routine-ontwerp bestaat uit twee taken: (1) het genereren de relaties tussen componenten in de vorm van netwerken en (2) het toe kennen vaan waarden aan onbekende parameters. De strategie voor de uitvoering van deze taken hangt af van de verdeling van de eisen en randvoorwaarden over de verschillende detailniveaus van het probleem, bijv. alleen globale eisen, alleen op het laagste detail niveau of als een mix. Deze verhouding is echter niet a priori bekend, en verschillend voor verschillende verdelingen van randvoorwaarden over het ontwerp. Van daar, dat de complexiteit in de routine ontwerp wordt veroorzaakt door de verdeling van de ontwerpeisen over de verschillende abstracties van het probleem. Het bepalen van de afhankelijkheid tussen ontwerp de complexiteit en de synthese strategien is de belangrijkste uitdaging van dit onderzoek. Het resultaat is een aanpak, de zogenaamde Computational Design Synthesis by Complexity Management, waarin zes methoden worden gentegreerd.

De eerste stap van deze aanpak is het formuleren van een ontwerpprobleem met de *FBS based Formulation* methode. Deze methode maakt gebruik van FBS representaties om ontwerpers te steunen bij het bepalen van de exacte componenten,

parameters, relaties en beperkingen die een rol spelen in het ontwerpprobleem.

Het geformuleerde probleem wordt vervolgens opgedeeld door het toepassen van de *ADT based Decomposition* methode. Dit resulteert in een nieuwe probleemformulering, bestaande uit verschillende niveaus van abstractie. De ontleding is zowel functioneel als fysiek en wordt beschreven met een structuur van functies, onderdelen, parameters en relaties uit het probleem.

Het ontlede model wordt vervolgens vertaald in een *Topology Abstraction Representation Diagram* (TARD). TARD bestaat uit vier bouwstenen: Elementen, C-relaties, H-relaties en ACO-relaties. Elementen vormen de afzonderlijke onderdelen van het ontwerpprobleem en groeperen de parametersets van het probleem. C- relaties modeleren de verbondenheid van de elementen in de topologie. H-relaties beschrijven hoe C-relaties zijn gerelateerd aan de samenstelling van een hoger niveau element. ACO-relaties worden gebruikt om analysemethoden, fysieke beperkingen en de samenhang met doelfuncties te modeleren. De twee belangrijkste kenmerken van TARD zijn: het ondersteunt de beschrijving van de verschillende eisenverdelingen van het ontwerpprobleem en helpt het definiren van top-down en bottom-up synthesestrategien.

De volgende stap in de aanpak is het bepalen van het *Topology System of Equations* (ToSE) van de TARD model. ToSE bestaat uit algebrasche vergelijkingen die bepalen hoe de instantiering van een component is gerelateerd aan de instantiering van andere componenten. Deze vergelijkingen modeleren de relaties tussen de componenten op n detailniveau (balansvergelijkingen), alsmede tussen de verschillende detailniveaus (verticale vergelijkingen) in een TARD model. Deze methode maakt het mogelijk om de topologie van het ontwerpprobleem te vertalen in vergelijkingen die vervolgens kunnen worden behandeld door bestaande constraint solving algoritmen.

Het bepalen van een synthesestrategie wordt gedaan door de toepassing van de *Local Grammar* methode en de *KGM Solving* algoritme. De eerste methode beschrijft hoe netwerken van componenten gegenereerd worden. Het tweede algoritme bepaalt hoe binnen verschillende abstractieniveaus verder wordt gegaan met het oplossen van ToSE. Bovendien bepaalt dit algoritme ook de volgorde waarin ontwerpparameters worden opgelost.

Verschillende voorbeelden beschrijven de toepassing van deze technieken en twee software-implementaties laten zien hoe deze technieken leiden tot de automatisering van ontwerpproblemen. De eerste implementatie automatiseert het ontwerpen van koelsystemen voor spuitgietmatrijzen. Door het combineren van de methoden met gespecialiseerde algoritmen, is de software in staat om automatisch koelsystemen te genereren voor een bepaalde matrijs. Een gebruikersinterface is ontwikkeld met de SolidWorks© API waarmee gebruikers in staat zijn om hun problemen te definiren door de geometrie in te voeren en de eigenschappen ervan. De tweede uitvoering is een gereedschapskist die implementeert *TARD*, *ToSE* , de *Local Grammar* methode en het oplossen van *KGM Solving* algoritme op een generieke manier. Na het invoeren van een TARD model van een bepaald ontwerpprobleem, is de toolbox geschikt om kandidaat-oplossingen te genereren.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CAS	Computer Aided Synthesis
DPD	Digital Product Development
CAD	Computer Aided Design
CAE	Computer Aided Engineering
FEA	Finite Element Analysis
CFD	Computational Fluid Dynamics
MES	Mechanical Event Simulations
CAM	Computer Aided Manufacturing
PDM	Product Data Management
CDS	Computational Design Synthesis
ADT	Axiomatic Design Theory
CDS	Computational Design Synthesis
FBS	Functional Behavior/State Structure
FBPSS	Function Behavior Principle State Structure
FRs	Functional Requirements
DPs	Design Parameters
CSIM	Cooling System design for Injection Molding
ATC	Advanced Technology Center
GA	Genetic Algorithms
PaRC	Parameters, Resolve rules and Constrain rules
ADT	Axiomatic Design Theory
TARD	Topology Abstraction Representational Diagram
ToSE	Topology System of Equations
DM	Design Matrix
CDS-CM	Computational Design Synthesis by Complexity Management//

Part I

Research Introduction

Chapter 1

Vision and Research Description

This thesis presents the results of researching the complexity of artifactual routine design problems. The main motivation is the future development of general purpose design automation systems. Design complexity is researched as a means of developing methods to automate design problems. This chapter introduces the research by describing its context, focus and scope. It finalizes with a brief description of the research approach.

1.1 Context: Computers in Engineering Design

From paperclips to digital folders, dwellings to skyscrapers, bicycles to airplanes; the world we live in is constantly being reshaped by design. But, what is it meant by design? First of all, depending on whether "design" is used as a noun or as a verb it gets different meanings. When used as a noun, different people understand it in different ways, like for example wallpapers, buildings, clothes, coffee machines or cars. But when it is used as a verb, most people would agree that it is a process of human creation. In fact, Bruce Archer [3] stated that *"design is a human activity concerned with the ability to mold the environment to suit material and spiritual needs"*. So, to design is a process. However, the characteristics of this process depend on what the purpose of design is. When designing a building, architectural design processes are used; while designing an airplane requires engineering design processes. Furthermore, designers do not only deal with the design of "new artifacts", but also with understating the rationales of their design processes. By doing so, improved design processes have emerged that are capable of designing better artifacts, more efficiently and using less resources.

1.1.1 The Engineering Design Process

Since the early '60s, the design community has made important improvements in the understanding and systematization of the engineering design process [52]. These have resulted in different theories and methodologies, which have enabled the design of artifacts as complex as spaceships and airplanes. The most accepted design methods nowadays are: (1) The Theory of Technical Systems by V. Hubka and E. Eder [25], (2) A Systematic Approach to Engineering Design by G. Pahl and W. Beitz [52], (3) Axiomatic Design Theory by N. Suh [72], (4) Product Design and Development by K. Ulrich and S. Eppinger [83], (5) The Mechanical Design Process by D. Ullman [81], (6) The General Design Theory by T. Tomiyama and H. Yoshikawa [79, 86] and (7) C-K Theory of Design by A. Hatchuel and B. Weil [24]. Figure 1.1(a) shows the design process according to Pahl and Beitz [52]. This process is divided into four main phases. In order to explain these phases, lets consider the design of a bicycle:

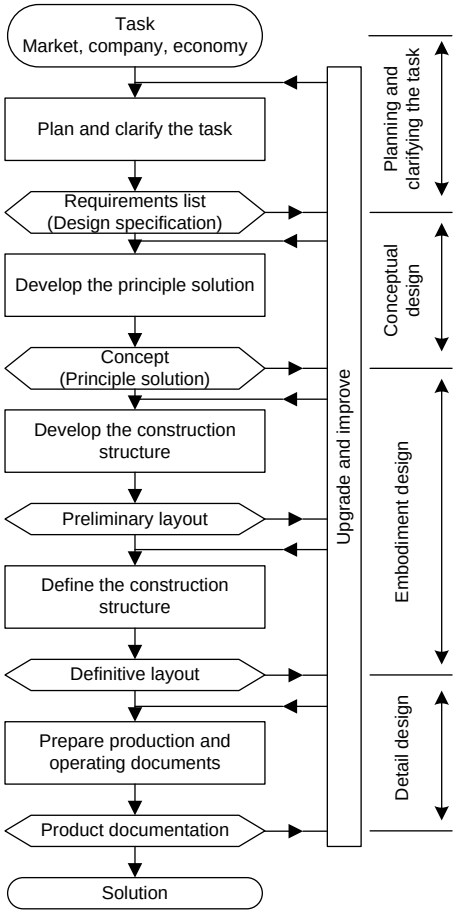
1. Planning and clarifying the task: A market study determines the preferences on the types of bicycles, colors, prices, costumers characteristics, etc. These preferences are transformed into a set product requirements: mountain bike, red, not larger than 2 meters, etc.

2. Conceptual design: The requirements are used to design a conceptual description that includes the components and principles of the bicycle. For example, the shape of its frame, the choice of having an electric engine, the number and shape of the seats.

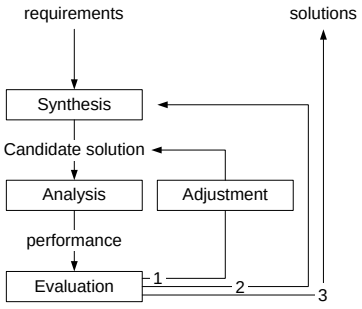
3. Embodiment design: The conceptual design is further detailed by choosing materials, the size of the wheels, the length and diameter of the bars, etc.

4. Detail design: The results are documented and manufacturing processes are planed. For the bicycle this means to determine the number and type of manufacturing machines, assembling order, etc.

The first phase of this process regards the problem statement, while the fourth phase aims at planning the manufacturing. So, both are organizational processes. The phases where the artifact is designed occurs in the second and the third phase, namely, the conceptual and embodiment design phases. Both phases are accomplished by following the processes depicted in Figure 1.1(b). This model, presented by Schotborgh et al. [59], shows that a ***synthesis*** process transforms the set of input requirements into a candidate solution. This solution is then ***analyzed*** to obtain measures of its performance. Resulting performances are ***evaluated***, to decide whether to modify (path 1), reject (path 2) or accept (path 3) the candidate solution. In case the quality of a candidate solution can be improved by small modifications, an ***adjustment*** process is followed. This thesis focuses on the synthesis process.



(a) The engineering design process according to Pahl and Beitz [52].



(b) Steps of the conceptual and embodiment design processes [59].

Figure 1.1: *Models of the design process*

1.1.2 The Role of Computers in Design

Ever since the emergence of computers, design researchers and practitioners have developed computer based systems to support engineering design tasks. With modern advances in technology, computers have become faster, more accessible and capable of handling increasing amounts of data, information and knowledge. This has transformed the design process, leading to the progressive substitution of paper based techniques by Digital Product Development (DPD) approaches. Nowadays, DPD is mainly supported by the following types of systems:

- Computer Aided Design (CAD): Is used to represent geometries and material properties of artifacts. Support the representation of conceptual and embodiment solutions that resulted from a synthesis process.
- Computer Aided Engineering (CAE): Supports the analysis of design solutions by simulating the artifact under working circumstances. Typical methods are based on Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD) and Mechanical Event Simulations (MES). In some cases, optimization tasks are also supported, which enables the adjustment process in Figure 1.1(b).
- Computer Aided Manufacturing (CAM): Assists the fourth phase of the design process, by supporting the development of manufacturing plans and process.
- Product Data Management (PDM): Supports the exchange and organization of information during the whole design process. Aids the communication between different design departments and keeps databases consistent.

These systems aid engineers in the design of complex products. However, advances in technology and competitive markets are driving modern products towards further miniaturization, better quality, more functionality and yet lower prices [45]; imposing a great challenge on product development. This has motivated the need for computer systems that support the synthesis process as well [4]. As a response, academia has researched and developed methods and tools to support the synthesis activity. In this context, the research presented in this thesis deals with the development of methods to automate the synthesis process of artifactual design.

1.2 Focus: Computer Aided Synthesis

In this thesis, Computer Aided Synthesis (CAS) is defined as software that automates, partially or entirely, the activity of design synthesis. The input of such systems are under-constrained design problems and its output a set of design

candidate solutions. The development of CAS systems is the result of the integration of different scientific disciplines, and its formal field of study and research is Computational Design Synthesis (CDS), presented in Chapter 2. This section describes the expected functionality of future CAS systems. This is done by presenting in subsection 1.2.1 a fictional story about the design of wind turbines with CAS. Subsection 1.2.2 describes some properties that CAS systems should have in order to reproduce this functionality. Subsection 1.2.3 positions this thesis in relation to the challenges posed by these properties on the development of CAS.

1.2.1 Story Board: Designing Wind Turbines with CAS

Wind Turbine Design

Wind turbines, as for example the one shown in Figure 1.2, are rotating machines that transform kinetic energy of the wind into electricity. The main components of a wind turbine are a tower, a rotor, a gear box, an electric generator, a controller, a brake and an anemometer. According to geographic characteristics and governmental regulations, wind turbines are designed using different materials, configurations and shapes. For example, a wind turbine to be placed off-shore (e.g. wind farms in the North Sea) requires a special coating material to prevent it from corrosion. On the other hand, its allowable noise levels are probably much higher than those of wind turbines placed in urban areas.

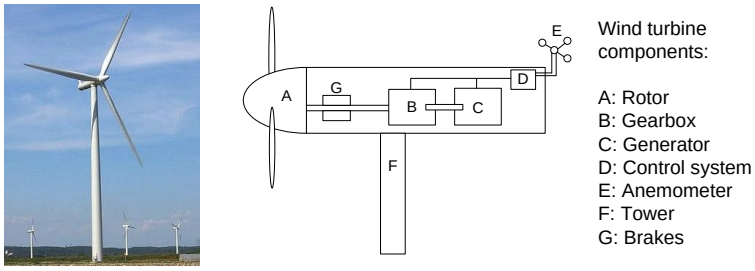


Figure 1.2: *Photo and topologic diagram of a wind turbine*

Design Session with CAS

A team of designers of an important energy company has a new project: design a wind turbine for skyscrapers. The two major specifications are to produce low noise levels and to minimize weight. Physical constraints regarding the turbine placement and assembly represent two of the main challenges of this project. The team has been using a special CAS system over the past years, which will enable the usage of libraries containing previously designed components and turbines.

The libraries also contain a knowledge base of wind turbine functions, behaviors and design rules.

The designers proceed by specifying the problem characteristics in their CAS system, as for example spatial, behavioral and manufacturing requirements. By accessing the CAS libraries, the designers are capable of selecting previously used functionalities and components they want to include in the new design. Once this is done, the system starts generating candidate solutions.

The CAS system generates solutions by applying different mechanisms. First, a design management routine recognizes the building blocks required for generating solutions, and decomposes the overall problem into problem chunks. The manager engine also decides in which order each problem should be solved and determines which expert algorithm should solve which problem. After this, expert algorithms continue by generating candidate solutions to their subproblems, which are later integrated to form the problem's overall candidate solutions.

Once a number of candidate solutions has been generated, the design team gathers around a screen and explores the automatically generated wind turbine designs. CAS has a special solution exploration tool to do this. The designers explore the performances of all generated solutions using a graphical solution explorer view. This tool is used to compare the different performances of different candidate solutions easily and relatively fast. The designers decide to select a number of solutions with low costs and low manufacturing efforts. A CAD module is then used to generate the 3D models and to allow the visualization of the turbines' geometry and configuration.

Although the solutions meet the initial requirements, the team of designers wants to explore the possibility of generating solutions based on different principles. For this purpose, the CAS system has an Internet based synthesis engine that searches a universal library of functions, behaviors and components. This library has been fed by thousands of designers around the world, and contains product independent design knowledge. As the designers are interested in innovative principles, they use as input an incomplete functional network. After waiting a couple of seconds, a number of solutions appear on screen. The found solutions are analyzed by both the computer and the designers. A large portion of the Internet found design principles are dismissed, leaving a few feasible solutions. These candidate solutions are further detailed by the CAS system in another design session. The solutions are compared to the ones generated in the previous session. Although the innovative solutions result in lower noise levels, its elevated cost and complexity persuade the design team to select one of the initially generated wind turbine concepts.

1.2.2 CAS Properties

This subsection describes some of the properties of CAS systems that would enable the functionality described in the previous story. This list is inspired in: (a) the research project Smart Synthesis Tools carried out by the University of Twente

and the University of Delft presented in [60], of which this research is part of; (b) the reflections "Intelligent computer-aided design systems: Past 20 years and future 20 years" presented by T. Tomiyama in [78] and (c) the paper presented by D. Ullman entitled "Toward the ideal mechanical engineering design support system" [82]. Although this list is not exhaustive, it provided the guidelines driving the research presented in this thesis.

Drawing your own problems

CAS systems should be capable of solving design problem structures rather than specific design problems cases. Instead of developing one system for individual design problems, there should be one system for design problem families. Furthermore, problems modeled in CAS should be stored in libraries for their reusability.

Domain knowledge independence

Generation strategies and algorithms have to be uncoupled from specific problem domain knowledge. However, there should be the possibility of using domain knowledge to steer the solution generation process. An example of the latter is presented by W. Schotborgh in [58].

Distribution of Requirements

CAS should generate solutions independent of the distributions of requirements in the problem, which implies:

- Configuration of the requirements: The system should be capable of handling different requirement configurations for a given specific problem. For example, when designing a compression spring, requirements can be set on a given wire diameter or on the spring constant or on both. In any case, the software should be capable of generating solutions.

- Requirements at different levels of detail: Setting requirements should be possible at different levels of detail. To illustrate this, lets consider the design of a computer. On the one hand, the user of a CAS system should be capable of defining the functionality network of the system, thus, the upper abstraction level. On the other hand, he/she should also be capable of defining the capacity of a Hard Disk Drive (HDD), thus, the lower abstraction levels.

Internet integration

Internet serves as a large pool of knowledge. By integrating CAS with Internet, these knowledge can be made available for the generation of design solutions. The first steps towards this goal have been achieved by the NIST Design Repository

Project [75]. NIST is a framework where component information in the form of functions, behaviors and flows can be stored.

Integration with existing systems

The successful implementation of future CAS systems requires a sound integration with existing CAD, CAE, CAM and PDM systems. As this integration will probably change the "classical" way in which designers approach their design processes, new types of User Interfaces (UI) have to be researched and developed. Two examples can be found in [61] and [69].

1.2.3 CAS Development Challenges

These properties pose a number of challenges on the development of CAS. The research presented in this thesis focuses on the following:

- Domain knowledge independence: Formalizing a generic framework for modeling design problems.
- Drawing your own problems: Developing standard building blocks for representing both specific components and specific behaviors.
- Distribution of Requirements: Determining the dependency between a synthesis process and the distribution of requirements.

1.3 Scope: Artifactual Routine Design

The scope of this thesis lies within the field of engineering artifactual routine design. This section explain what is meant by this at the hand of the FBS model.

1.3.1 FBS model

FBS models a design artifact by distinguishing the following levels of object representation: Function, Behavior/State and Structure, as shown in Figure 1.3. The basis of the FBS model is that the transition from function to structure is performed via the synthesis of physical behaviors. Therefore, behavior allows characterizing the implementation of a function. As many different views of the FBS model have been developed and researched, this thesis adopts the unified FBPSS model presented by Zhang [87]. This model is based on the analysis and generalization of the Japanese [84, 85], European [52], American [11] and Australian [19] schools of design modeling. The FBPSS model uses the following definitions:

- **Structure**: Is a set of entities and relations among entities connected in a meaningful way. Entities are perceived in the form of their attributes

when the system is in operation. For example, in Figure 1.3 the Structure is represented by an electric motor and a crank mechanism. Here, the two possible entities (structures) are the lengths of the bars L_1 and L_2.

- **States**: Are quantities (numerical or categorical) of the Behavioral domain (e.g. heat transfer, fluid dynamics, psychology). States change with respect to time, implying the dynamics of the system. For example, in Figure 1.3, the states of the structure are represented by the distance L_0 between the electric motor and the piston, the torque T of the electric motor, or the displacement of the piston s.

- **Principle**: Is the fundamental law that allows the development of a quantitative relation of the States variables. It governs Behavior as the relationships among a set of State variables. For the example in Figure 1.3, two possible principles are electromagnetism ruling the operation of the electric motor, and solid mechanics ruling the function of the crank mechanism.

- **Behavior**: Represents the response of the structure when it receives stimuli. Since the Structure is represented by States and Structure variables, Behaviors are quantified by the values of these variables. In the case presented in Figure 1.3, the two Behaviors are *Generate torque* and *Convert torque into force*.

- **Function**: It is about the context sensitive usefulness of a system for its existence. For example, in Figure 1.3, one possible function of this system is *to compress gas*.

1.3.2 Classification of Design Problem

If one considers a design artifact as an object with a complete FBS description, a design problem can be defined as one with an incomplete set of descriptions. Different classifications of design problems can be formulated from the FBS model. In this thesis, the classification is chosen according to the types of incomplete representations and according to the types of behavior. As shown in Figure 1.4, according to the types of incomplete representations design is classified in:

- ***Routine design***: One in which the space of functions, behaviors and structures is known, and the problem consists of instantiating structure variables.

- ***Innovative design***: One in which the functions and behaviors are known, and the design consist of generating new structures that satisfy them.

- ***Creative design***: One in which the functions are known, and the problem consists in determining the structures and behaviors required to satisfy them.

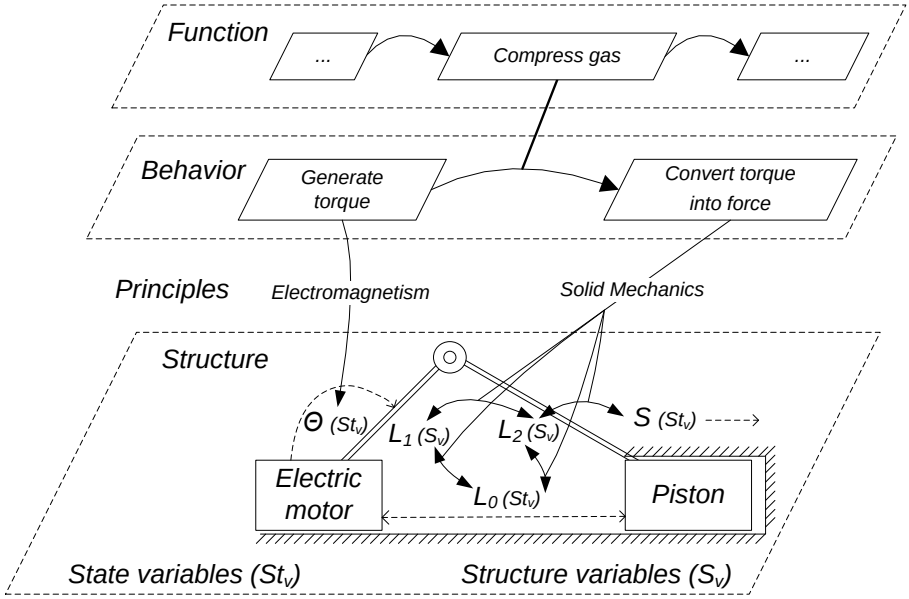


Figure 1.3: *FBS example of a crank compression mechanism.*

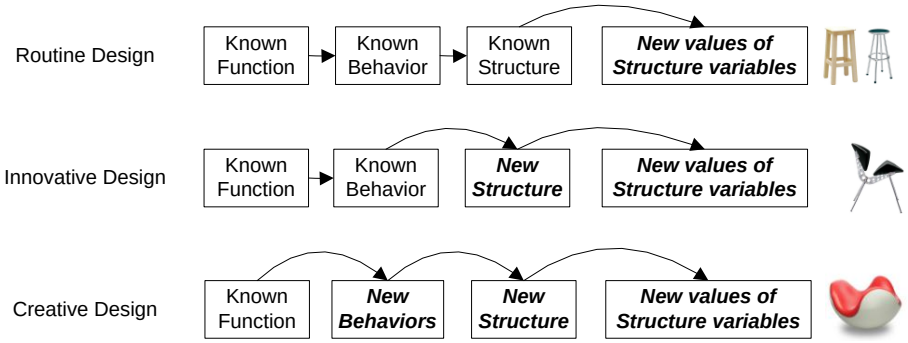


Figure 1.4: *Design problem classification.*

Nature encompasses a vast variety of behaviors (physical, chemical, human, etc). Considering physical and human behaviors, design can be classified in:

- ***Engineering design***: Behaviors are characterized by principles stated in the laws of physics. Depending on the discipline of study, engineering design can be further classified into mechanical, electrical, chemical, geological, etc.
- ***Human centered design***: behaviors are characterized by physiologic, psychological and emotional human reactions. Two examples are architectural design and industrial design.

Under these definitions, the scope of this thesis lies within the boundaries of engineering routine design problems. Furthermore, emphasis is set on problems composed of parametric and topologic models, as it will be described in Chapter 3.

1.4 Vision: Bottom-up Approach to CAS

Figure 1.5 shows how routine, innovative and creative design are performed within different dimensions of design representations. The figure also shows that innovative design encompasses routine design, and creative design encompasses both innovative and routine design. From this perspective, understanding the rationales of creative design requires the previous understanding of innovative design, and likewise the understanding of innovative design requires the previous understanding of routine design. For the development of CAS, this means that before assisting creative and innovate design activities, first a sound comprehension of the automation of routine design needs to be developed. Moreover, C. Wynne states in [12] that *"routineness is an individuals standard, measured in the brain of the beholder"*. In other words, what one designer perceives as creative, another more experienced designer regards as routine. Therefore, the routineness of a design problem depends on the available knowledge designers have on functions, behaviors and structures. In this sense, it is expected that having libraries of routine design problems will enable, after further research, the automation of more innovative and creative problems.

Although CDS in routine design has been broadly researched in academia [12], most methods have been developed for specific applications [4]. Moreover, Cagan et al. [4] stated that the initialization of CDS has not received enough attention in literature, and that this might be the reason why it has not been widely implemented in industry. While it is true that advances in CDS have enabled the development of design automation methods for specific routine design problems, few methods describe how to do so from a general perspective [4]. Therefore, this work investigates the development of generic methods to automate the synthesis process of routine design problems.

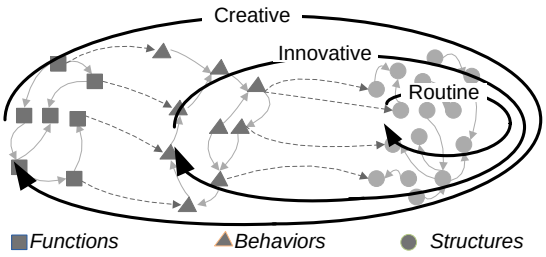


Figure 1.5: *Bottom-up approach to computational synthesis research.*

1.5 Challenge: Complexity in Routine Design

Although routine design occurs within a well defined domain of knowledge, several industrial cases clearly demonstrate the complexity that such problems can exhibit. Consider for example the design of injection molds. The first injection molds were designed and developed in 1868 by John Wesley Hyatt, who injected hot cellulose into a mold for producing billiard balls [15]. Much later, in 1946, James Hendry built the first screw injection molding machine, giving birth to the machines and processes we know nowadays. Since then, much knowledge on injection mold design has emerged and been formalized in books (e.g. [15, 44]), expert systems (e.g. [46, 10]) and Internet. However, given the amount of components, physical phenomena and processes involved, the design of injection molds is still considered a complex task. As one may imagine, automating the synthesis process of mold design, though being routine, is not straightforward. Given that around 80% of design at industry is routine [43], a proper understanding of its complexity is a relevant topic in the field of design theory and methodology.

1.5.1 Modeling Design Artifacts

Artifacts, e.g. an injection mold, can be modeled as a hierarchical multi-layered network of interrelated components and parameters that resemble the structure of the pyramid of Gerrit Muller [48], as shown in Figure 1.6. In this model, the top layers represent functional requirements, the in-between levels represent components, and the lower levels represent design parameters of these components. Functional requirements specify the characteristics of an artifact's function, as for example the power of an electric engine. Furthermore, in this model components are composed of networks of other sub-components, and so forth. For example, sliders in injection molds are composed of mechanical linkages, which are simultaneously composed of rigid links and joints. It is characteristic to complex artifacts to have a large number of interconnected networks of components, as well as a large number of parameters, relations and constraints.

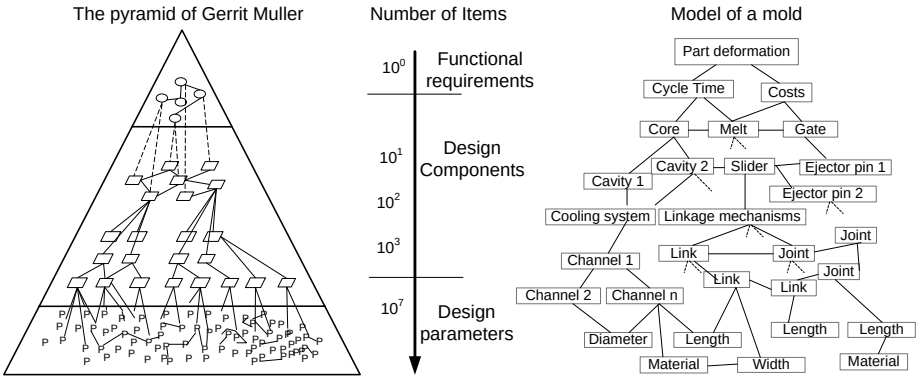


Figure 1.6: *The pyramid of Gerrit Muller [48] to model artifacts*

1.5.2 Modeling Design Problems

An artifactual design problem can be modeled as an incomplete description of an artifact, as it is shown in Figure 1.7. The descriptions known on forehand are regarded as the design requirements, and these must be satisfied by candidate solutions. Design requirements can be functional requirements, components, parameter values or combinations thereof. Creative, innovative and routine design problems can be represented using this model, as the differences among them reside in the amount and type of knowledge available for generating candidate solutions.

In routine design there is knowledge available about:

- the types of components that can be used to generate candidate solutions,
- how components are allowed to be connected among each others,
- parametric descriptions of each component, and
- relations and constraints that relate parameters and components to functional requirements.

Furthermore, designing one type of artifact can be the subject of different types of problems, as several combinations of design requirements can be formulated.

1.5.3 Synthesis in Routine Design

As Figure 1.8 indicates, moving from an incomplete representation to a complete description is done by a synthesis process. Synthesis processes in routine design are performed by two types of tasks: (1) generating networks of components and (2) attributing values to unknown parameters. The exact strategy determining how these tasks are performed depends on the distribution of requirements

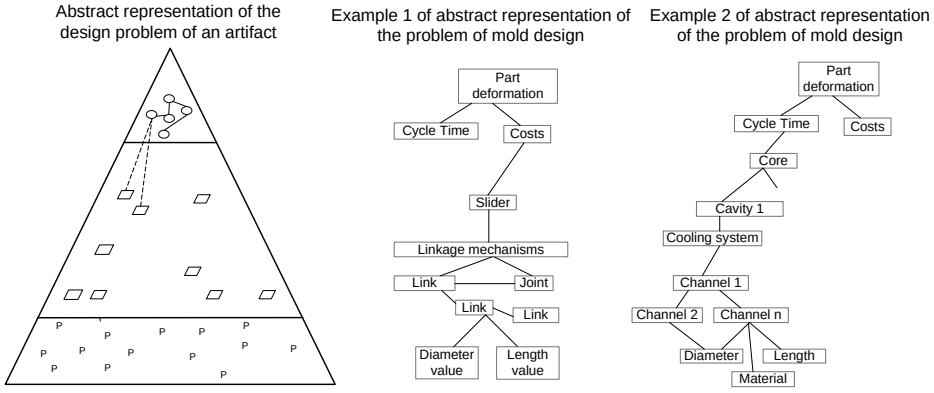


Figure 1.7: *Modeling design problems as incomplete representations of artifacts*

throughout the different levels of detail of the problem, e.g. only on top, only at the bottom or as a mix. However, this relation is not known a priori and is different for different distributions of design requirements.

1.5.4 Complexity in Routine Design Problems

Complexity in routine design problems is attributed to the distribution of design requirements along the different abstractions of the problem. Determining the dependency between design complexity and a synthesis strategy is the challenge this research deals with.

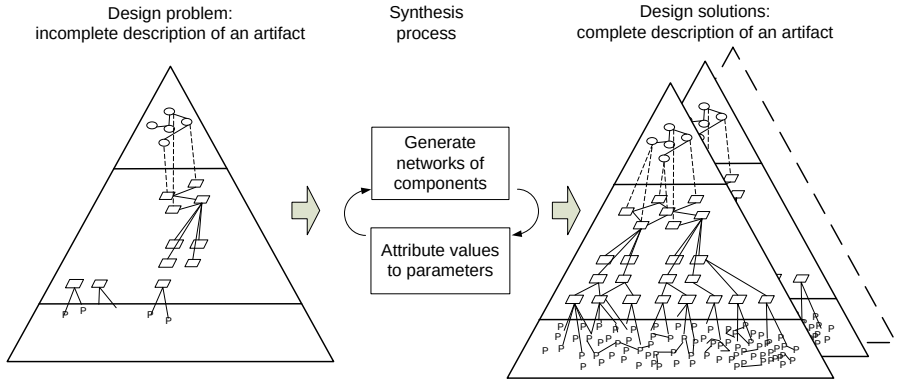


Figure 1.8: *Synthesis: from incomplete to complete descriptions*

1.6 Research: Managing Complexity In Routine Design

The work presented in this thesis proposes the management of complexity as means of determining synthesis strategies to solve routine design problems. The research approach consisted in: identifying different types of complexity in routine design, developing methods to manage each type of complexity, and integrating the methods into one methodology that determines the synthesis strategy for solving a routine design problem. By implementing the resulting methodology into computer programs, routine design problems are automated.

1.6.1 Hypothesis

The research presented in this thesis is founded on three hypothesis:

A routine design problem can be characterized by a finite number of complexity types.

A method can be found to manage each complexity type.

A generic method to determine the synthesis strategy of routine design problems can be found by determining the types of complexity in the problem and applying methods for their management.

This under the assumption that:

- complexity in routine design is related to the distribution of design requirements throughout the different levels of detail of the problem, and
- synthesis strategies depend on the complexity of the problem.

The results of investigating the first hypothesis are presented in Part I. The result is a model of complexity in routine design. Part II presents methods that were developed during this research for managing the previously identified types of complexities. Part III integrates these complexity management methods into one methodology for determining synthesis strategies for routine design problems.

1.6.2 Case Study

The research presented in this thesis is part of the project Smart Synthesis Tools being developed at the University of Twente in cooperation with Delft University of Technology, both in The Netherlands. The project researches the development process of CAS systems. The aim is to deliver generic development methodologies for dedicated synthesis tools supporting engineering design processes [60]. The case study assigned to this research project is the design of **C**ooling **S**ystems for **I**njection **M**olding (CSIM). The Advanced Technology Center (ATC) department of PHILIPS has provided the knowledge about this design problem. Chapter 2 presents a brief description of the characteristics CSIM design.

1.6.3 Complexity Management Approach

The approach proposed in this thesis for managing design complexity, as means of determining synthesis strategies for routine design problems, consists of the following steps:

1. Determine a formal model of the components, parameters and relations of the design problem to automate. This is done by applying the FBS based design formulation method described in Chapter 5.

2. Decompose the design problem into different levels of detail by analyzing its functional and physical structure. The ADT based design decomposition method described in Chapter 5 is proposed for this end.

3. Translate the obtained problem model into a TARD representation. TARD representations are maps that describe which components, relations and parameters can be used for designing an artifact. Chapter 6 describes the rationals of TARD.

4. Determine the Topology System of Equations (ToSE) of the model, following the method in Chapter 7. ToSE are algebraic relations that determine how the instantiation of one component is constrained by the instantiation of other components. These equations relate components within one level of detail (balance equations), and between different levels of detail (vertical equations).

5. Apply a constraint solving algorithm (Chapter 8) and a grammar generation method (Chapter 7) to determine the design's problem synthesis strategy. The constraint solving algorithm determines which components to instantiate by solving the ToSE equations. The grammar method further specifies how to generate a network of components. Furthermore, the constraint solving algorithm also determines the order in which the design parameters are solved.

The last two steps of this approach have been implemented into software applications. One tool is an specific software implementation for the design problem of cooling systems for injection molding. This tool is capable of determining synthesis strategies, which in combination with other algorithms, allows the automatic generation of cooling systems. The second implementation is rather generic. Here, a designer enters as input a TARD model of a routine design problem and the system, in combination with a random generation algorithm, generates candidate solutions. The effectiveness of this implementation can be improved by using better algorithms.

Chapter 2

Research Positioning

The following review is a collection of accepted literature associated with the proposed research into complexity management for routine design automation. Its purpose is to establish a background for the reader regarding the areas of computational synthesis and complexity in design. It also shows how previous research is incorporated in this thesis. A short description of CSIM design is also presented.

2.1 Field: Computational Design Synthesis

Research in Computational Design Synthesis (CDS) studies algorithmic procedures to automate the generation of designs [2]. The idea is that by combining "low-level" building blocks, "high level" functionalities can be achieved. CDS methods vary from straight forward implementation of artificial-intelligence (e.g. [56]), constraint solving (e.g. [33, 16]) and optimization techniques (e.g. [89]) down to much more specialized approaches, as for example engineering shape grammars ([41]) and function-based synthesis methods ([70]). CDS is a multidisciplinary science integrating knowledge from diverse disciplines, among which:

- Design theory, cognitive science and artificial intelligence: have developed prescriptive and descriptive methods to model design processes; have resulted in better understandings of the types of design problems, reasoning approaches, problem structures, design rationales and representations.
- Optimization, constraint solving and operations research: have provided strategies, algorithms and methods for the automatic generation and evaluation of design solutions.
- Knowledge engineering: has allowed the usage of knowledge to structure design problems and aid the process of synthesis. Other contributions are

knowledge elicitation techniques, knowledge representation methods and frameworks, as well as knowledge based expert systems.

- Computer science: has provided software languages, information models, algorithmic procedures and practical techniques for the implementation of computer systems.

The following subsections describe how these disciplines have been integrated into methods for CDS.

2.1.1 General Method

A CDS method describes models and algorithms required for automating a given synthesis process. Figure 2.1 shows a flowchart with the general processes a CDS method should incorporate [8]. Firstly, the design problem is formulated. For technical problems this is done by declaring variables, relations, constraints and objective functions. This information is translated into representations, or building blocks, that can be used by algorithms to generate candidate solutions. A candidate solution satisfies all constraints of the problem independent of how well the goal is achieved. An evaluation step analyses the solutions by calculating its performance and decides whether it is accepted, adjusted or rejected. Guidance drives the generation process in a given direction to improve the generated solutions.

CDS methods range from low level building blocks manipulation up to high level conceptual reasoning. This thesis deals with the first type of methods. In relation to this flowchart, this thesis focuses on the formulation (Chapter 3), representation (Chapter 4, 5 and 6) and generation (Chapter 7 and 8) phases. The following subsections present a review of CDS methods relevant to the topics treated in this research. A complete overview on design automation methods and techniques can be found in Chakrabarti [9] and in Antonsson and Cagan [2].

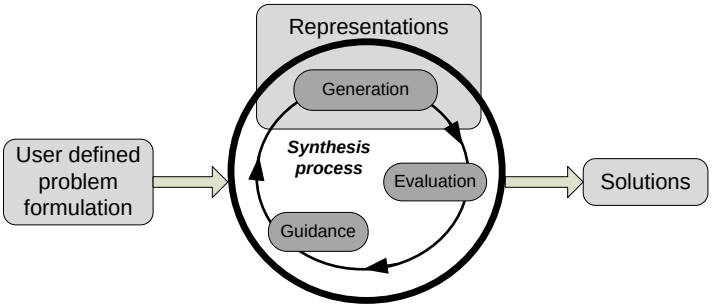


Figure 2.1: *Computational design synthesis diagram [8].*

2.1.2 Grammars

Using grammars for CDS consists in translating knowledge about a design problem (e.g. from experienced designers) into a set of transformation rules describing how an initial incomplete design can be transformed into a complete one. Here, algorithms generate solutions by applying the rules to an initial design. Grammars are used both as representations (e.g. [54, 51]) and as generation systems (e.g. [35, 64, 68, 71]. As generative systems, grammars have been typically used in architecture (e.g. [17]) and in mechanical design (e.g. [68]).

A grammar is defined by a 4-tuple $G = (V, X, R, S)$, where V is the set of objects that are manipulated by the grammar, X is a set of terminal and non-terminal symbols, S is the initial symbol and R is a set of rules of the form outlined above. The language of the grammar G is the set of all results produced from the start symbol that consists of only terminal symbols.

Figure 2.2 shows a set of exemplary transformation rules referring to shape grammars for a planar truss design [63]. Accordingly, the first rule of the illustration states that a given triangular truss can be divided into two triangles, whereas the second rule states the inverse transformation. Rule 3 applies on a slightly different structure as rule 1 (a triangle with a fixed point) and consequently proposes an appropriate transformation, while rule 4 also allows this transformation in a reverse fashion. A structure composed of multiple triangular truss features can successively be transformed by applying these rules.

There are three possible tasks for programs that implement grammars [20]. The most common task, and perhaps the first that comes to mind, is to aid in the generation of designs at the hand of a known grammar. Here, grammar rules are combined using special algorithms to generate a new design. A second type of program is a parsing program. A parsing program is given a grammar and a design, and the program determines if the design is in the language generated by the grammar and, if so, gives the sequence of rules that produces the shape. A third type of program is an inference program. The grammatical inference problem is: given a set of designs, construct a grammar that generates the design solutions.

This thesis researches the applicability of grammar approaches as function of the type of requirements and available resources in a design problem. A generative approach based on grammars in presented in Chapter 7.

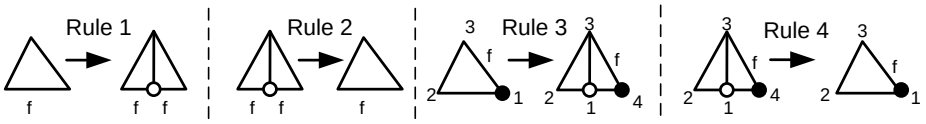


Figure 2.2: *Grammar of truss design.*

2.1.3 Agent Based Design

Agent based design is inspired in how multiple members of a design team contribute to generate design solutions. Here, individual designers (agents) work individually in their specific problems coordinated by a manager that keeps control of the overall design process. In agent based design, components are synthesized based on the physical interactions between them [76].

Agents are meta-heuristics which have been used to solve a wide variety of optimization and constraint satisfaction problems [47]. It is based in encapsulating software modules into agents which can then be organized into teams. The agents collaborate with each other by communicating results via shared memories. A-Teams can handle multiple objectives and constraints naturally, and do not require a set of weighting functions a-priori, allowing the user to select from a set of pareto equivalent solutions at the end of the solution process.

A-design, proposed by M. Campbell et al. in [7], is a CDS method in which agents modify design candidates and are themselves modified in order to create better results. The basic subsystems of the A-Design theory are (1) an agent architecture that is responsible for creating and improving design alternatives, (2) a representation of the conceptual design problem that is comprehended by the agents in order to create design concepts, (3) a scheme for multi-objective decision making that retains solutions exhibiting different patterns of strengths and weaknesses in order to accommodate change in user preferences, and (4) an evaluation-based iterative algorithm for improving basic design concepts towards successful solutions.

M. Campbell et al. present in [6] the application of this theory to the configuration problem of electro-mechanical devices. As shown in Figure 2.3, it uses four types of computational agents. ***C-agents*** construct conceptual candidate solutions using a library of component types. ***I-agents*** generate conceptual solutions with values obtained from a catalog of components. Each I-agent has a different preference, e.g. low costs of high performance. ***M-agents*** receive good designs and produce feedback to control the other agents in the system. ***F-agents*** also receive good designs, but their function is to modify solutions with expectation of improving them.

The solutions generated by I-agents are classified into three groups: poor designs, good designs and pareto designs. Poor designs are discarded by an evaluation algorithm. Good designs are modified by M- and F-agents. Pareto designs are added to a "hit list" and simultaneously used by modification agents in the search of better designs. Each iteration step yields a new "best" design. After the iterations ends, the "best of the best" is presented to the user as the solution.

From the perspective of A-Design, this thesis sets the focus on the development of a manager agent. The goal is to have one generic manager agent which can determine synthesis strategies for routine design in general.

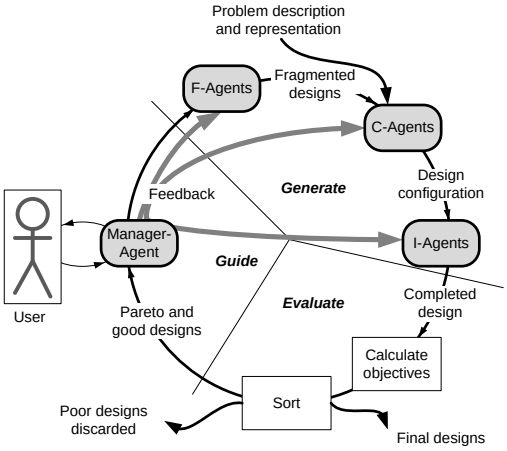


Figure 2.3: *CDS and agents in A-Design [7].*

2.1.4 Evolutionary Approaches

Evolutionary approaches are based on the principles of biological evolution, and encompass a broad range of search techniques, among which the most prevalent are genetic algorithms (e.g. [32]), evolutionary programming (e.g. [18]) and evolution strategies (e.g. [88]).

The search algorithms of evolutionary approaches are based on genetic adaptive operators, namely, mutation and recombination. The selection is based on Darwinian selection, or selection of the fittest. A characteristic of these methods is that they employ sets of solutions (populations) represented by vectors. Any particular solution is written as a string of objects called chromosomes, and each component is regarded as a gene. All three methods consist of maintaining a population of competing candidate solutions. The first population of solutions is randomly generated. Then, new populations are created by bringing random alteration to the solutions in the previous population. The performance of a solution determines how fit it is. A selection mechanism determines which solutions to combine for generating new solutions, and which ones to exclude.

The main differences between these techniques are their representation and selection methods, which are:

- Genetic algorithms: The representation is done in the form of string of numbers, which are often binary. The selection is done by calculating a fitness based on an analysis.

- Genetic programming: uses lisp-like trees as representation of the solutions, resulting in computer program like solutions. The fitness is determined by the ability of the generated program to solve a computational problem.

- Evolutionary programming: the structure of the program is fixed, being the parameters the ones allowed to change. Evolutionary programming uses vectors of real numbers for the representation of solutions. The selection has the characteristic that the results of mutations are considered in future generations only if their fitness is higher than in other found solutions.

Evolutionary approaches incorporate the processes of Synthesis, Analysis, Evaluation and Adjustment required in CDS as it is shown in Figure 2.4. This characteristic makes the implementations of such approaches relatively straightforward for CDS.

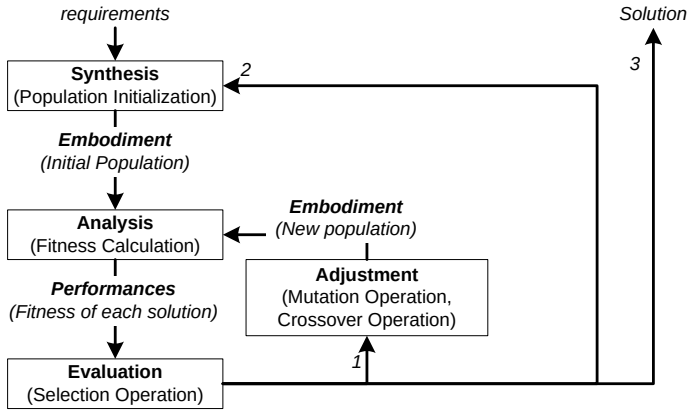


Figure 2.4: *Genetic algorithm and the computational synthesis method.*

2.1.5 PaRC

PaRC (acronym for Parameters, Resolved rules and Constrain rules) is a knowledge engineering methodology for design automation [57]. PaRC prescribes the automation of parametric routine design problem by proposing a framework for representing knowledge and methods for automating the design problem. The major characteristic of PaRC in relation to other CDS methodologies, is that it covers the path from a knowledge source (e.g. an expert with a design problem) up to the development of software tools (see Figure 2.5)

PaRC distinguishes two types of design knowledge: (1) knowledge that defines what the design problem is, and (2) knowledge that defines how the design problem can be solved. Design problem knowledge is represented by parameters and topological elements. Knowledge for solving the problem is represented by resolve rules (R-rules) and constraint rules (C-rules). A generation algorithm, generic to design problem formulated with the PaRC representation, enables the synthesis

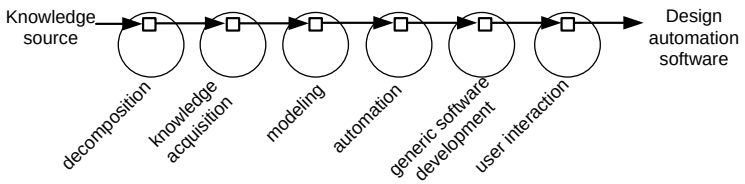


Figure 2.5: *PaRC: Knowledge engineering method, from [57].*

activity. PaRC prescribes two methods for translating a parametric routine design problem into software for automatic design generation. On the one hand, a knowledge elicitation method targets the acquisition of problem and problem solving knowledge. On the other hand, a method for developing the software tool is presented. Software development is supported by a generic software framework that facilitates the codifications of new design automation tools.

In relation to PaRC, the long term goal of this research is to automate the development of CDS methods at the hand of the structure of the problem. The results of PaRC have served in envisioning the development process of computational synthesis systems. Furthermore, this thesis builds up the model of information classification presented in PaRC.

2.2 The Problem: Complexity Management

Complexity in design has been studied from two different perspectives: the physical domain and the functional domain [73]. In the physical domain -which includes most engineers, physicists and mathematicians- complexity is seen as an inherent characteristic of physical things, including algorithms, products, processes, and manufacturing systems. This kind of thinking leads to the idea that systems with many parts are inherently more complex than those with less. In the functional domain, complexity is seen as a relative concept that evaluates how well we can satisfy "what we want to achieve" with "what is achievable". This view is used in this thesis to develop a model of complexity for routine design problems.

This section aims at presenting a short overview of studies on design complexity that are the most relevant to this work. From the functional perspective, the theories of axiomatic design, incompleteness of information and multi-disciplinary complexity are described. From the physical perspective, complexity in large parametric spaces is described.

Complexity in design can be attributed to the different stages of the design process. In this sense, there is complexity of synthesis, complexity of analysis, complexity of evaluations, complexity of the design solutions, complexity of managing the design process. Here, I will only present complexity of the synthesis process.

2.2.1 Complexity in Axiomatic Design

ADT is based on the hypothesis that there are fundamental principles that govern good designs [72]. Its two founding axioms are:

1. Maintain the independence of the Functional Requirements (FRs)
2. Minimize the information of the Design Parameters (DPs).

FRs are the set of requirements that characterize the needs of the artifact in the functional domain, while DPs are the variables that characterize the design in the physical domain. The relation between the FRs and the DPs is represented in equation form as:

$$FR = [A]DP \tag{2.1}$$

where A is the Design Matrix (DM) of the problem. Depending on the DM, a design can be coupled, decoupled or uncoupled. Consider for example a problem with two FRs and two DPs. When the design is coupled, the FRs cannot be satisfied independently because of the interdependence with both DPs, as shown in Equation 2.2. In a decoupled design, shown in Equation 2.3, the DPs have to be solved in a particular order so that FRs are achieved. In uncoupled designs (Equation 2.4), the FRs are independent from each others, and no particular order is required for solving the DPS.

$$Coupled: \quad \begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} x & x \\ x & x \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \tag{2.2}$$

$$Decoupled: \begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} x & 0 \\ x & x \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \tag{2.3}$$

$$Uncoupled \quad \begin{Bmatrix} FR1 \\ FR2 \end{Bmatrix} = \begin{bmatrix} x & 0 \\ 0 & x \end{bmatrix} \begin{Bmatrix} DP1 \\ DP2 \end{Bmatrix} \tag{2.4}$$

In ADT, complexity is defined as "the measure of uncertainty in achieving the functional requirements of a system within their specified design range". When the range of a system changes as function of time, it is regarded as a system with time-dependent complexity. When the range does not change as function of time, it has a time-independent complexity. "Time" is used in a general sense, signifying progression of "events".

Time-independent complexity is classified into time-independent real complexity and time-independent imaginary complexity. The former is a consequence of the system range not being inside the design range. The latter occurs when there are many FRs and the design is a decoupled design. It is called imaginary because this corresponds to a situation in which the different orders in solving the design matrix have different attributed levels of difficulty. A system with imaginary complexity can satisfy the FRs at all times if we vary DPs in the right order.

Time-dependent complexity is the uncertainty caused by the increase or decrease of the number and types of DPs during the design process itself. ADT classifies time-dependent complexity into combinatorial and periodic complexity. Design problems with combinatorial complexity experience a continued growth of their DPs in time. For example, constructing a sentence by the combination of words has combinatorial complexity. As the number of words (the DPs in this case) increases, keeping semantic and syntactic consistency among them becomes more difficult. On the other hand, periodic complexity is the case in which the increase of parameters is restarted after a period of time (or succession of actions). An example described in [74] is air traffic control. Air traffic in large airports follows wave pattern that depends on the time of the day. When the traffic is at its peak, air controllers deal with very complex situations. However, at low traffic times their task becomes significantly more simple.

ADT suggests three main strategies for managing design complexity:

- Minimize the number of FRs
- Eliminate time-independent real complexity and time-independent imaginary complexity,
- Transform a system with time-dependent combinatorial complexity into a system with time dependent periodic complexity

This thesis adopts this model of design complexity, and explores its characteristics for routine design problems. The developed complexity management methods are based on the three previously suggested strategies.

2.2.2 Completeness of Information

As presented by Ueda [80], complexity of emergent systems can be categorized in three classes with respect to incompleteness of information about the artifacts environment and/or the specification of the problem. Figure 2.6 shows how the complexity of a design problem increases as function of the incompleteness of information for different domain.

The three classes pointed out in [80] are:

- Class 1: Problems with complete descriptions. Are those where the environment information and the problem descriptions are given wholly. In this case it is often difficult to find the optimal solution.
- Class 2: Problems with incomplete environment descriptions. The difficulty in these problems is characterized by uncertainties in modeling the dynamic behavior of the system.
- Class 3: Problem with incomplete specifications. Environment and problem specifications are both incomplete. Difficulty resides in the ambiguity of the problem, having human interaction as the most significant aspects.

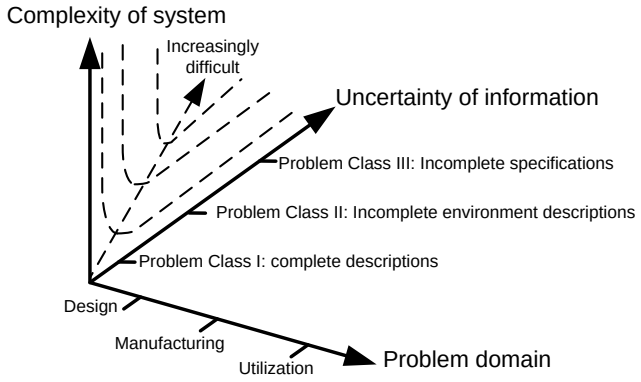


Figure 2.6: *Complexity as function of knowledge completeness, from [80].*

In this research the focus is set on complexity class 2. For this type of problems, there are many candidate solutions, which lead to a combinatorial explosion of the space of solutions. Here, Ueda has pointed out the need to develop efficient and robust methods to find optimal solutions. Typical problem solvers here are evolutionary algorithms, genetic programming methods, and genetic algorithm, among others. The preference on choosing one technique or another depends on the characteristics of the problem, e.g., purely parametric, topologies, lay outing, etc [27]. For complex routine design, where the problems usually present a mixed combination of characteristics, collaborative usage of different techniques is preferred. The difficulty here lies in the proper management of techniques and solutions though the generation process.

2.2.3 Complexity of Multi-disciplinarity

Tomiyama [77, 78] researched design complexity resulting from the involvement of multiple disciplines. He examined the structure of knowledge represented by relationships among theories. Consider the design of an electromechanical engine, where the mechanism dynamics and electromagnetism are integrated by the multi-disciplinary character of the artifact. In this case, two independent theories become relevant to the instantiation of a variable -e.g. length, width, depth- common to both theories, as indicated in Figure 2.7. These type of complexity is regarded as Complexity by Design, being commonly found in multidisciplinary design problems (MDDP). Intrinsic complexity of multidisciplinary is also identified as a second type of complexity arising from MDDP. Here, two theories get integrated through one common concept as an interface between them, as shown in Figure 2.7.

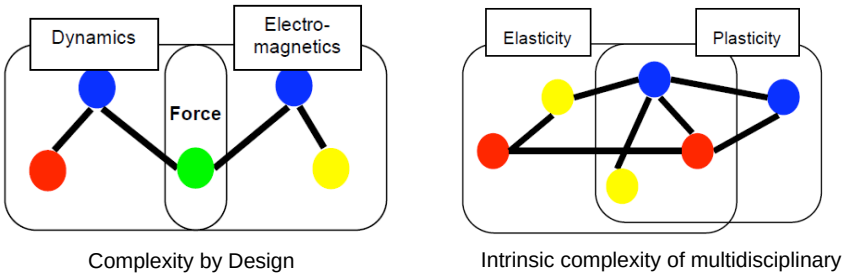


Figure 2.7: *Complexity from the viewpoint of knowledge structure [78].*

2.2.4 Large Parametric Spaces

An important source of complexity in design is attributed to large amounts of inter-constrained parameters. To cope with these types of complexity, design problems are usually partitioned into smaller and easier to solve sub problems, which are latter solved towards optimal solutions in a coordinated fashion. One category of decomposition methods is Multidisciplinary Design Optimization (MDO) [53]. Here, systems are partitioned along discipline boundaries, which are often dictated by the design organization structure or available analysis tools. The focus lies in dividing the optimization into the sub-optimizations within each module and a coordinating optimization at the system level place.

MDO methods can be subdivided into single-level and multilevel. The former centralizes all design decision in one optimizer, while the latter distribute decision making throughout the system by providing design groups with some autonomy and ability to utilize existing design tools. Notable multilevel MDO techniques are Collaborative Optimization (CO) [1] and Bi-Level Integrated System Synthesis (BLISS) [66]. Another category of decomposition methods is Analytic Target Cascading (ATC) [1]. Here, the problem is partitioned in a hierarchical form around the components composing it. The method consists in specifying top level targets (performances to achieve) and cascading the targets to the lower level components design.

In order to use these methods, it is necessary to already have a first best guess initial design, which is refined to produce optimal designs. However, these types of methods is applicable for redesign proposes rather than for CDS of new solutions. Therefore, this type of complexity is not regarded in this study. As it will be discussed in Chapter 4, this type of complexity is rather attributed to the solution space than to the problem space.

2.3 Case Study: CSIM Design

Injection molding is an important manufacturing technique for producing plastic parts. This technique consists of injecting a hot polymer into the impression of a mold. Figure 2.8 shows an exploited view of a mold. A mold is composed of two basic parts, namely, a core and a cavity. Here, the plastic is cooled down to a solid state by a series of cooling channels drilled into the mold. The cooling stage is of great importance in the injection molding process, as it affects the productivity and the quality of the final part.

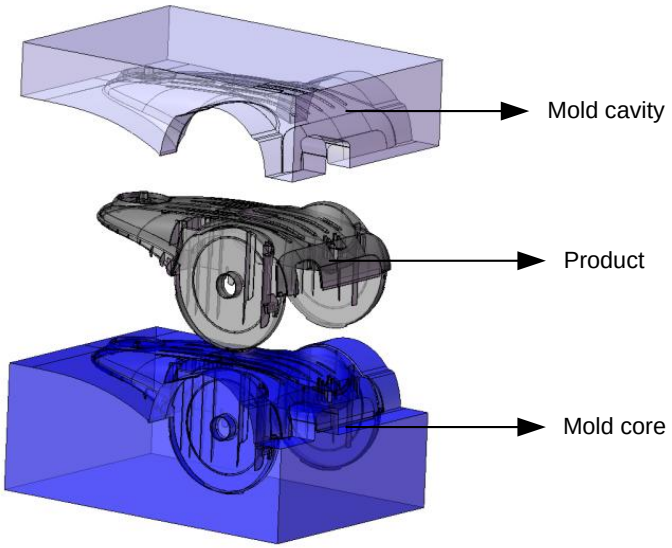


Figure 2.8: *Injection mold example.*

2.3.1 Cooling Design

Cooling systems for injection molding are defined as the set of components that enable heat transfer between a hot melt contained in a mold and a coolant substance flowing within it. Cooling systems allow reducing the time required for solidifying the plastic melt, which is translated in manufacturing cost reduction. In addition, cooling systems should be capable of delivering uniform temperatures on the plastic part so that unwanted effects, such as differential shrinkage and warpage, can be minimized. A cooling system for injection molding is composed of a temperature-controlling unit, a pump, coolant supply manifold, hoses, cooling channels, and collection manifold. For the present case study, the cooling

channels layout design has been selected from the other components, as it determines to a great extend the performance of the cooling system. Figure 2.9 shows an example of a cooling layout. Sliders, ejector pins and other mold moving parts constraining the mold volume make it hardly feasible to design optimal cooling layouts.

The design process of CSIM is divided into three successive processes [39]:

1. Preliminary design: consists of determining the possible locations of cooling channels in the vicinities of hot spots. At this phase, the channels do not form a circuit.

2. Layout design: consists in connecting previously positioned channels into circuits to assure a coolant is able to flow through it. Inlet and outlet channels are included to exchange the coolant with external cooling devices.

3. Detailed design: Optimizes cooling channels position to minimize warpage and thermal residual stress by applying small changes to the channels positions.

In this research, the preliminary and layout design phases are used as case study, as it is in these phases where the cooling concepts are generated.

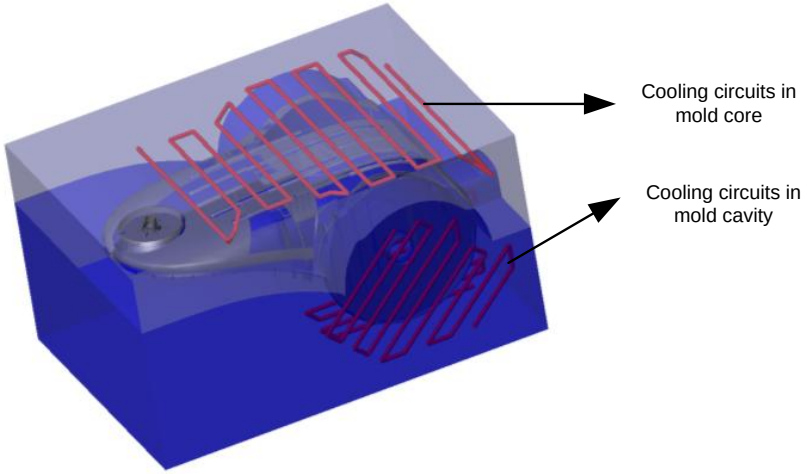


Figure 2.9: *Example of a cooling System for injection molding.*

2.3.2 Related Work

At present, most academic research in injection molding cooling design has focused on developing detailed analysis and optimization strategies. The former simulate the cooling phase of the injection molding process and has resulted in Computer Aided Engineering (CAE) systems as MoldFlow [67] and Moldex3D [40]. The latter uses heat transfer analysis to minimize cooling time and temperature distributions of the plastic part. However, a human designer has to create the initial design from where CAE and optimization methods can be applied. In fact, optimizing cooling channels placement and diameter results in local optimum, given that the initial design limits the possibility of finding global optima. Therefore, in order to achieve complete automation, the problem of generating the initial design must be solved.

Although automating design problems has been extensively investigated over the past 30 years [37], not much literature addresses the problem of injection molding cooling design. The most relevant to this thesis is the research developed by Li et al. [37, 39, 38], which has resulted in a method for automating the design of cooling circuits for injection molding. In short, the method consists of decomposing the part geometry into several predefined shapes. Then, three techniques are collaboratively used for candidate solution generation: (1) case-based design, (2) graph-based search and (3) heuristic search. Case-based design maps the shape features to predefined solutions, obtaining a preliminary design that is captured in a graph model. A graph based transversal algorithm is employed to search for candidate cooling circuits. Heuristic search develops the candidate solutions into layout designs that contemplate tentative manufacturing plans. Although this approach has demonstrated to be capable of automating the generation of cooling solutions, it suffers from the drawback that the design has a strong dependency on the accuracy of the shape recognition algorithm as well as on the quality of the sub-solution predefined for each shape feature. Furthermore, complex algorithms are required to solve geometric constraints and keep the physical consistency of the cooling solutions (e.g. make sure cooling channels are connected to form a circuit).

Part II

Founding Frameworks

Chapter 3

Information and Models

This chapter explores the entities and types of information in artifactual design required to formulate routine problems. As the focus is computational synthesis, mathematical models and relations used in design automation are presented. The chapter finishes by presenting a classification of design problems as function of the types of information.

3.1 Introduction

Design artifacts are described by three different types of entities: (a) vocabulary of elements, (b) descriptions of elements and (c) the configuration of elements. Consider the case of the gear device shown in Figure 3.1. Here, the vocabulary of elements is two gears and two shafts. Descriptions determine the attributes of the artifact, like the diameter (D) and angular velocity (w). Configurations determine the disposition of the elements in the structure, as for example the connectedness between elements represented by the relations in the figure. Configurations can be classified into topologic relations and physical coherence constraints. While the former define the topology of the elements in the structure, the latter is used to assure no physical impossibilities are committed by the artifact being designed. For example, two gears cannot share the same place in space.

The following sections describe types of information in routine design problems and a scheme for their formulation. This scheme is used in Chapter 4 to identify different types of complexity.

3.2 Types of Information

As proposed by Schotborgh et al [59], the information flow of an analysis and synthesis process can be classified in three groups: embodiment, scenario and performance, as shown in Figure 3.2. According to this model, analysis allows the

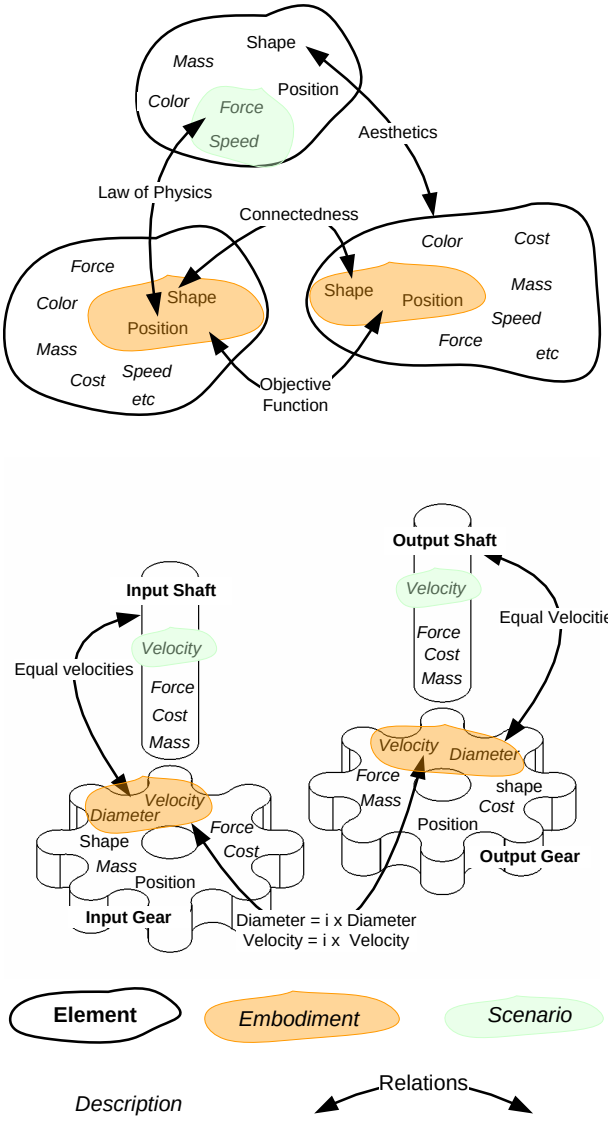
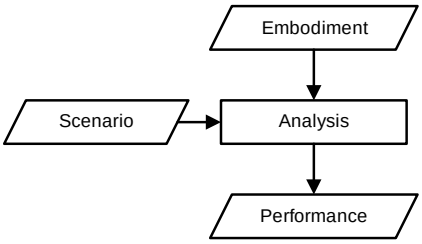
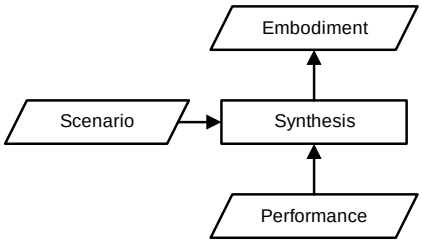


Figure 3.1: *Types of information in routine design.*

quantification of the performances of an embodiment undergoing a given scenario (Figure 3.2(a)), while synthesis consists of specifying an embodiment undergoing a given scenario such that performances are satisfied (Figure 3.2(b)). Thus, two opposing processes handling the same types of information.



(a) Information flow in analysis process, from [59].



(b) Information flow in synthesis process information

Figure 3.2: *Information flow in analysis and synthesis processes*

Embodiment

Embodiment is here defined as the subset of descriptions of an artifact upon which instances are created to generate design solutions. In Figure 3.1 this is shown for the case of the gear device, where the embodiment is composed of two elements (one input gear and one output gear), and its descriptions (diameter and velocity).

Scenario

An artifact's ability to accomplish its function is affected by its interaction with its environment. The subset of environment variables, attributed to elements in the natural world and considered in measuring a design artifact's ability to accomplish its function, is here defined as scenario. Scenarios are also specified by descriptions. For the case of the gear device in Figure 3.2, a possible scenario is one shaft attached to the input gear and one connected to the output gear.

Performance and Goals

Given that design functions are expressed in abstract terms, it is necessary to use measurable descriptions to express and asses its goals. Goals are commonly represented by objective functions, where performance indicators are weighted and added to compute the overall performance of the design. Performances are calculated by analysis relations using instantiated embodiments and scenarios. Analysis relations use known principles -as for example laws of physics and economics-

to model the interaction of the design artifact with its environment and predict its behavior.

Goals can be *expressed* by defining requirements on the objective functions and performance parameters and *assessed* by calculating -using analysis relations- the performances and objective function of an instantiated artifact. Two types of goals are often found in design problems: constraint satisfaction and optimization. In constraint satisfaction, the objective is finding instances of design artifacts within the allowed topology relations and confinement constraints. Here, performances are used as means of assessing the behavior of the design. For optimization, maximizing or minimizing performances is added to the constraint satisfaction problem. Performances are used to express the desired quality of the design artifact.

Analysis in design is often done analytically, by simulation, or by a combination of both. They can vary from algebraic equations to complex differential equations. Finite element methods, computational fluid dynamics, circuit simulation, and other computational analysis tools offer accurate and robust analyses [4]. For the case of the gear device in Figure 3.1, a performance indicator could be the rotational speed of the output shaft. The objective function could be expressed by this performance, reducing the goal to that of the output rotational speed. The goal can then be expressed by a required rotational speed, while it can be assessed by an analysis technique for an instantiated design.

Design Rules

Design rules determine values of embodiment descriptions as function of scenario and performances descriptions. These rules are the result of experience, and constrain the ranges of permitted values parameters. Design rules aid the generation, evaluation and guidance processes by providing shortcuts to values that have proven to be successful.

3.3 Problem Formulation

Deriving methods and schemes for formulating design problems has been an important subject of research in the field of Problem Solving Theory (PST) [65, 22, 21] as well as in the field of CDS [57, 5]. From the perspective of PST, Simon [65] has defined problem structuring as the process of drawing upon our knowledge to compensate for missing information, and using this knowledge to construct the problem space. Here, design problems whose problem space are completely defined are regarded as well-defined, while those which are not are regarded as ill-defined. Well-defined problems can be solved using generally applicable problem-solving mechanisms, whereas ill-defined problems require other approaches [21]. As routine design problems proceed within a known space of functions, expected behaviors and structure variables and the problem is one of

instantiating structure variables [19], it is regarded as well-defined. According to PST [22], well-defined problems are formulated as function of a known initial state, a clear goal state, a constrained set of logical states and constraint parameters. Using this formulation structure, the following scheme is proposed for formulating routine design problems:

- ***Embodiment*** and ***scenario*** to describe the initial state of the design.
- ***Objective function***, ***performance indicators*** and ***analysis relations*** to express and assess the goal of the design artifact.
- ***Topological relations*** and ***physical coherence constraints*** indicate the set of logical states that have to hold for the design artifact to exist.
- ***Confinement constraints*** indicating the values embodiment, scenario and performance descriptions are allowed to get.

This scheme is used by the knowledge elicitation method presented by Olthof et al. [50]. The method allows identifying and formalizing routine design problems at industrial settings.

3.3.1 Example: CSIM Design

Figure 3.3 graphically represents the problem formulation of CSIM design by representing elements by nodes and relations by arcs. Labels specify the models of the elements and relations. For explanatory reasons, not all descriptions and relations have been included.

In the figure, Channel is regarded as embodiment element, while Mold Part and Plastic Part are regarded as scenario elements. The goal of the design, expressed by the objective function, is to minimize the cooling time and to minimize the temperature differences in the plastic part. Cooling time and temperature distribution are therefore regarded as performances. Analysis is represented by the equation of Laplace, and allows the calculation of the performances (temperature distributions and cooling time). Detailed information on the analysis methods can be found in [62]. Topology relations specify that Channel elements are connected to each other inside the mold and are not allowed to share its space with the Plastic Part element. Physical coherence constraints are used to define: (a) the minimum distance between Channel and Plastic Part (PCC-1), (b) the minimum distance between Channel (PCC-2), (c) the diameter values of the Channel (PCC-3) and (d) the allowed distance between Channels and the Mold Parts (PCC-4). In [44], knowledge from expert designers has been used to formulate these three quantities as a function of the plastic part thickness. Furthermore, other physical coherence constraints, such as non-drillable surfaces and inlet/outlet surfaces, are also described. As the model is used for explanatory reasons, the latter have been kept out of the formulation.

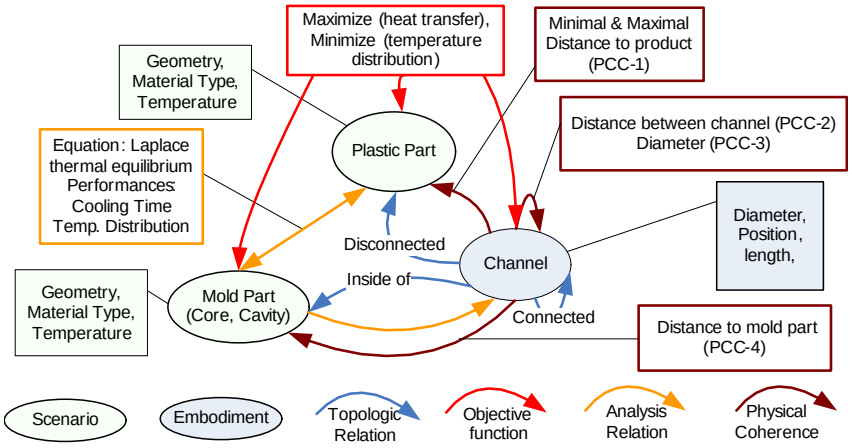


Figure 3.3: *Problem formulation of CSIM design.*

3.4 Models in Artifactual Routine Design

This section presents models to represent descriptions and relations used in the computational synthesis of artifactual routine design. The aim is to classify the information contents by using common data models, which can be regarded as basic building blocks for formulating artifactual routine design problems. Doing so facilitates the development of abstract formulations from where to differentiate families of design problems, and later develop algorithms to automate them.

3.4.1 Models of Descriptions

Design descriptions have been classified in five model categories: parametric, fields, space, shapes and topologies. Each category represents a different problem domain, as different problem solving approaches are used to generate solutions. In this sense, design problems comprehending several dimension have higher degrees of complexity and require a combination of methods for its automation. The work presented in this thesis mainly focuses on the domains *Parametric* and *Topology*.

Parametric

Parameters model properties valid for the entire element. They are used to represent attributes as material properties, color, weight, density, etc. These can be of different nature, as for example numeric, symbolic, logic, predicate, and combinations among them. For numeric parameters, confinement constraints define a continuous or discrete space of possible values. For symbolic, predicate and logic

ones, all possible values have to be specified. That is for every A_i $=(A_1 ... A_n)$ exists an A_i such that:

$$A_i \in (N, Z, Q, R, C) \tag{3.1}$$
$$A_i = \{true, false\} \tag{3.2}$$
$$A_i = Symbol \tag{3.3}$$

Spaces

Describe positional attributes of elements in a topology. Positional descriptions depend on the chosen coordinate system -Cartesian, Cylindrical of Spherical- and the dimensions of interest -1D, 2D, or 3D. These can be considered as numeric parameters related by a model that is determined by the chosen coordinate system. Values can either be continuous or discrete.

Fields

Use parameters and geometric vectors to describe properties that are valid in specific regions of the elements. Fields are specified together with an incident zone, shown in the Figure 3.4 as cubic. Incident zone is the spatial place where the field influences an element. An incident zone can be a volume, an area, a line or a point. Meshed CAD models are used in Computer Aided Engineering (CAE) software to specify incident zones. Vectors and parameters can then be attributed to each mesh-element. In Figure 3.4 an example illustrates how a parameters and vectors are related with their incident zones

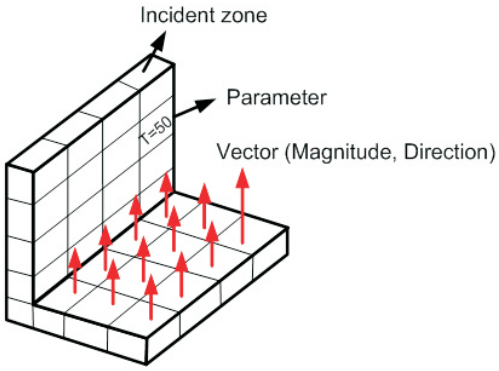
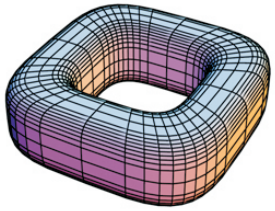


Figure 3.4: *Example of a field attribute.*

Shapes

Describe the form of the elements -or groups of elements- present in the design structure. Commonly used models are based on geometric relations and graphs. Geometry models the form by means of mathematic equations. Models are defined with parameters related by geometric functions, the latest being usually algebraic or differential. When the geometric function is known, parameters are instantiated to produce new shapes. When not known, the problem becomes one of finding the correct geometric relations. Polynomials are often used for this purpose, having the polynomial degree and its coefficients as unknowns. Depending on the case, confinement constraints are set either to parameters or geometric functions. Furthermore, perimeters, areas and volumes can also be subject of restriction.

Super quadrics [26] have been broadly used for this purpose. Super quadrics are polynomials whose degrees and coefficients values vary depending on the shape being modeled. In Figure 3.5 a toroid super quadric shape is shown together with its polynomial equation. By changing the values of the polynomial coefficient, the shape of the toroid can be changed.



$$(r-a)^2 = \left(\frac{z}{a_3}\right)^2 = 1$$

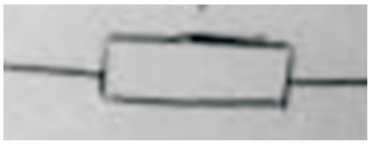
$$r = \sqrt{\left(\frac{x}{a_1}\right)^2 + \left(\frac{y}{a_2}\right)^2}$$

Figure 3.5: *Super quadric of a toroid, from [26].*

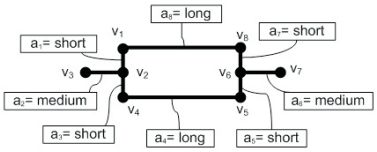
Attributed Graphs model shapes using nodes (representing shape primitives) and arcs (representing the connection within primitives). Primitives might be further decomposed into sub graphs, obtaining shape models with several levels of abstraction. When the shape graph is specified, design generation consist in instantiating attributes related to the graph's nodes and arcs. When the graph is not specified, but its primitives and relations are, shapes are generated by constructing new graphs. Confinement constraints can be set to both, the number and the types of primitives and relations. In Figure 3.6, a handmade symbol of an electrical resistor shape is modeled using such an attributed graph. Arcs of the graph represent segments, while the nodes are used to represent vertices. Labels are used to further specify the arcs.

Topologies

For design problems whose elements can be instantiated several times, its cardinality can be defined as a topology description. It reflects the number of instances



(a) Handmade resistor symbol.



(b) Graph representation of resistor symbol.

Figure 3.6: *Example shape graph of electric resistor symbol.*

of one element in the artifact. This variable allows controlling the instantiation of elements when the problem is one of generating topologic structures. The concept of cardinality also applies to topology relations. In this case, the number of allowable relations is determined by its cardinality.

3.4.2 Models of Relations

Different types of models are used in design to describe relations. Their characteristics determine the approaches required to handle them, and are here restricted to three basic types: algebraic, differential and logic models. However, this set is not restrictive as others can also be considered.

Physical Coherence and Objective Functions

Are commonly modeled with algebraic relations. In some cases, when the descriptions are symbolic rather than numeric, logic models can be assembled. First order logic and propositional logic models are the most common types of logic models in design.

Analysis

These relations are often modeled by a combination of all three types. Analytic methods are usually a combination of algebraic and logic models, while simulations make use of numeric methods to solve differential equations.

Topology

Topology is often modeled with set theory. One framework for doing so is Region Connection Calculus (RCC) [13], which in a qualitative fashion describes the relation among euclidean or topological regions. It allows the formalization of concepts as convergence, connectedness and continuity.

3.5 Common Design Problem Formulations

This section presents a classification of routine design problems families. The classification, shown in Table 3.1, is done by relating the type of descriptions and relations, presented in Section 3.4 to problem formulation scheme presented in Section 3.3.

Table 3.1: *Common artifactual routine design problems.*

Information Content		***P***		***C***		***L***		***Sh***		***T***	
Descriptions	*Model*	*E*	*S*	*E*	*S*	*E*	*S*	*E*	*S*	*E*	*S*
Parametric		+	+	+	+	-	-	+	+	+	+
Space		+	+	-	+	+	+	+	+	+	+
Field		-	-	-	+	+	+	-	+	+	-
Shape	Geometry	+	+	-	+	+	+	-	-	+	+
	Polynomial	-	-	-	-	-	-	+	+	-	-
	Graph	-	-	-	-	-	-	+	+	-	-
Topology	Elements	V		V		F		F		V	
Cardinality	Relations	F		V		V		F		F	

P = Parametric, C = Configuration, L = Layout
Sh = Shape, T = Topology Optimization
(+) = Present, (-) = Absent, (F) = Fixed, (V) = Variable

3.5.1 Parametric Design

When the design problem does not exhibit complex spatial, topologic and shape requirements; and all possible solutions adhere to a common template, it is possible to simplify the problem by modeling the artifact by a set of parameters (see Table 3.1). In these cases, problem solving becomes the process of assigning values to parameters in accordance with the requirements, constraints, and optimization criterion. At present, several algorithms exist for solving this type of problems, as for instance Genetic Algorithms (GA), Simulated Annealing (SA), Evolutionary Algorithms (EA), etc.

3.5.2 Configuration Design

For design problems that can be modeled in terms of predefined design elements and known topologic relation, the design process consists of assembling and configuring design elements. In this case, shape descriptions and spatial descriptions are not the main subject of design, as shown in Table 3.1 . Solutions need to satisfy design requirements and constraints, as well as to approximate some typically cost-related optimization criterion. Configurations can be generated by either instantiating new relation types, or by generating new elements in the topology. Grammatical approaches are very common in the generation of configurations.

Startling et al [71] developed a parallel grammar for design synthesis of mechanic clocks. A FBS design model of the clock was produced to map the possible Functions to embodiment Structures. A Function grammar (defining the connectivity between Functions) and a Structure grammar (based in the topologic relations of the clock) are used simultaneously to generate solutions.

3.5.3 Layout Design

Determining placement locations for components within a product housing or container is, in short, the goal of layout design. The embodiment elements are described by geometric relations and spatial attributes, while scenario elements impose constraints. It is characteristic to layout design to have multiple local optima, space discontinuities, high number of components, constraints, and multiple objectives; which makes it a difficult problem to solve. In Cagan et al. [4] references are given to different approaches for solving these types of problem.

3.5.4 Shaping

Consists of determining the shape of the embodiment elements. Solutions are generated by either defining new mathematic relations or by assembling new graphs structures. Shape grammars have been successfully used in the generation of elements shapes, as reported in [63]. They consist of construction rules that determine how shape primitives can be bounded to produce new shapes. McCormack et al. [42] developed the Buick Grammar, used to generate novel Buick forms. Super quadrics are used in [39] to recognize shape features in the CS of cooling systems for injection molding.

3.5.5 Topology Generation

Topology generation problems consist of determining the possible material arrangements in a design domain. This is often done by iteratively eliminating and redistributing matrial chunks until the specified structural performance is achieved. Solving this type of problems is often done by element-based methods, such as the homogenization or the SIMP method [89]. In such problems, embodiment is represented by field elements, while the scenario is represented by vectors or shapes.

Chapter 4

Design Structure and Complexity

This Chapter presents a framework to structure routine design problems and develops a model of its complexity. The results are positioned in relation to the model of complexity of Axiomatic Design Theory.

4.1 Introduction

In design, structure and complexity are two closely related concepts. Complexity is a property related to the degree of difficulty or uncertainty for finding a solution to a design problem. In order to analyze design complexity, it is necessary to understand the structure of the problem. The structure of a design problem has two important aspects to be studied:

- The distribution of design parameters along the problem model: all parameters concentrated in one element vs. several parameters scattered thorough several elements; one level of detail vs. several levels of detail.
- The relation between what is known (design requirements) and what is unknown: scattered along the problem model, concentrated in problem chunks, at the top (only functional requirements), at the bottom (only design parameters) or as mix of all these possibilities.

From this perspective, design complexity depends on the structure of the problem. This chapter describes different types of complexity that have been identified routine design problem structures. This has been done by relating ADT's complexity theory to the design formulation scheme presented in Chapter 3 and the structuring framework presented in Section 4.2.1 of this chapter.

4.2 Structuring Routine Design Problems

Physicists model natural phenomena through differential and integral equations. Specific problems are solved by setting boundary conditions on the differential and integral equations, and applying solving procedures to obtain analytical expressions. The resulting expression can then be used to calculate values of variables by specifying the values of the input parameters. Consider for example the law of heat conduction shown in equation 4.1. This differential equation models the phenomena of heat transfer through matter from a region of high temperature to a region of a low temperature. As this is done independent of geometry, material properties or temperature distributions, the equation is generic. To model a specific case of heat transfer the equation is rearranged by introducing boundary conditions, canceling unnecessary terms and performing mathematical manipulations. For example, heat conduction in one dimension between two flat plates results, after rearranging equation 4.1, in equation 4.2. The obtained equation can now be used to introduce known values and calculate the values of the required parameters, as for example in equation 4.3 the time required to get temperature $T = \psi$ at a point $x = \zeta$ is $t = \xi$.

$$\frac{\delta T}{\delta t} = a \frac{\delta^2 T}{\delta x^2} \tag{4.1}$$

$$t\left(x, T\right) = \frac{x^2}{\pi^2 \cdot a} \cdot ln\left(\frac{8}{\pi^2} \cdot \frac{T - T_W}{T_{MO} - T_W}\right) \tag{4.2}$$

$$t\left(x = \zeta, T = \psi\right) = \xi \tag{4.3}$$

If generalizing this structure, it can be noticed that physicist model natural phenomena at the hand of tree phases which are solved by two types of procedures, as it is shown in Figure 4.1. On the one hand, the three phases are: the differential/integral equation, the analytical expression, and the solution. On the other hand, the procedures are: differential and integral calculus to transform the differential equation (phase 1) into an analytical expression (phase 2), and algebra or numeric methods to transform the analytical expression (phase 2) into values of unknown parameters (phase 3).

From a research perspective, this problem structure has allowed:

- defining the types of representations required for modeling each domain (i.e. operators),
- studying the complexity of each domain by analyzing the configuration of the used representations (i.e. equation order, equation linearity),
- developing procedures and operations to solve each problem domain (i.e. Laplace Transformations, Newton-Raphson method)

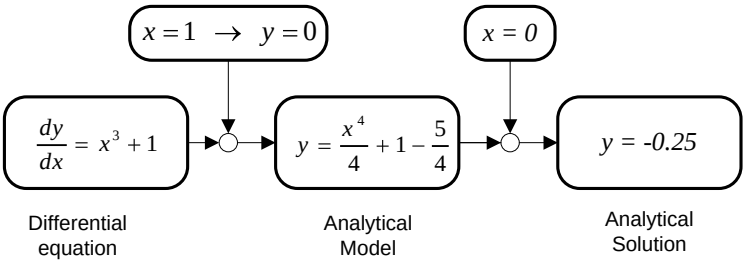


Figure 4.1: *Structure of problems in modeling natural phenomena.*

- identifying common problem structures (i.e. Poisson's equation, Laplace equation)

On the other hand, some of the major advantages from an application perspective are:

- reusing existing problem formulation,
- utilization of standard solving methods,
- development of computer based simulation tools.

Structuring design problems as done in natural sciences is likely to drive the automation of design problems toward more generic approaches, with advantages in both research and application. For this research, such an structure would allow the identification of features causing complexity, and do so independent from the problem semantics. Furthermore, strategies for managing design complexity can be formulated as function of problem structures.

4.2.1 Structuring Framework

By making an analogy between the information contents presented in Chapter 3 and the structure described in the previous section, it is proposed to structure routine design problems at three different abstraction: problem class, problem instance and problem solution. This is shown in Figure 4.2. Problem classes are transformed into problem instances by specifying its requirements. Requirements are specified by instantiating descriptions. Problem instances are transformed into problem solutions by algorithms that generate instances to the unknown descriptions. Therefore, one problem class can represent many problem instances, and one problem instance can have many problem solutions, as indicated in Figure 4.3. Under this view, solving routine design is analogues to solving problems with known differential equations.

This structure can also be used to structure innovative and creative design problems. Solving innovative design is analogous to combining different differential equations to model various interrelated physical phenomena. Solving creative design problems is analogous to developing new differential equations.

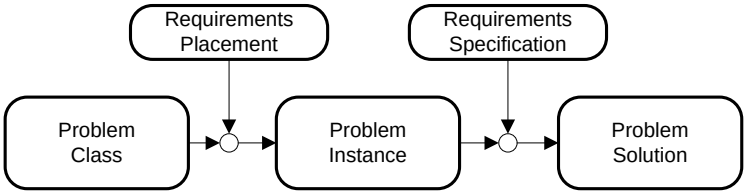


Figure 4.2: *Framework to structure design problem.*

Problem Class

A problem class is structured in:

- Elements: are considered class descriptions, and are used to represent both, embodiment and scenario elements. As it will be described in Chapter 5, Element can also be differentiated by assessing the functions and types of descriptors modeling a component.
- Relations: are considered class descriptions and are of different types, namely, topology, physical coherence, design rules, analysis relations and objective functions. Their descriptions can be declared within the scope of the relation or by pointing towards descriptions of embodiment and scenario elements.
- Descriptions: are variables that characterize elements and relations by mathematic models.

Problem Instance

A problem instance is structured by:

- Instantiated scenario: represent scenario specifications,
- Partially instantiated embodiment elements or parameters: represent embodiment requirements, and impose constraints to the space of possible solutions.
- Instantiated performance parameters: represent the performance specifications the embodiment has to meet.

Problem Solution

Consist of fully instantiated elements, relations and parameters. For underconstrained problem-instances, many solutions may exist for one problem instance. This depends on how constrained the problem is. An underconstrained will allow for multiple solutions, while a systems of equation type of problem will have a limited number of solutions.

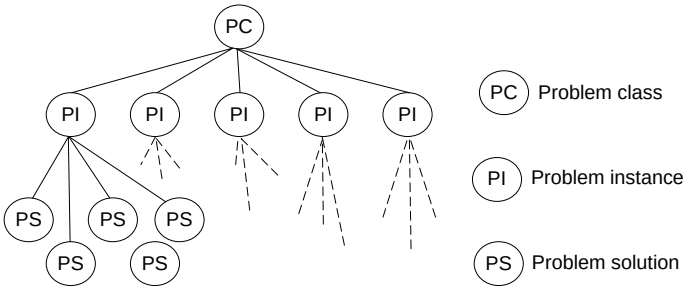
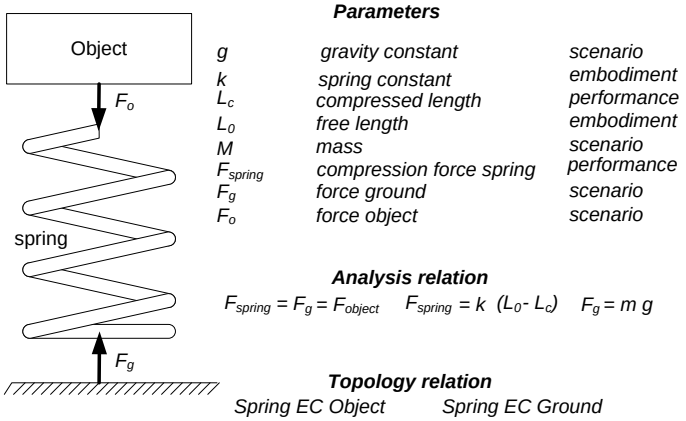


Figure 4.3: *Problem structure dependencies.*

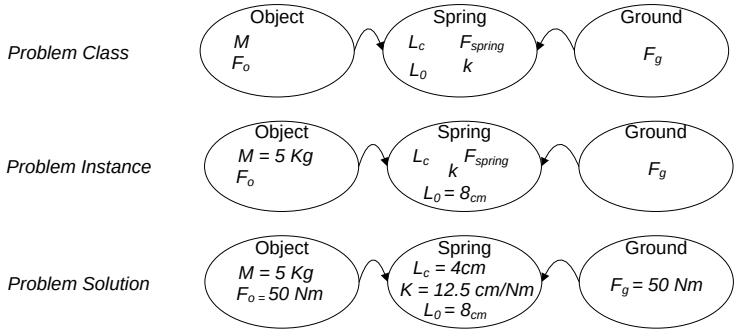
The mayor consequence of utilizing this structure in the development of CAS systems, is that generation algorithms and software architectures have to be capable of handling different types of problem instances and problem classes. Developing methods for automating specific routine design problems have to be valid for a design problem class, rather than for specific problem instance.

4.2.2 Example: Spring Design

Consider for example the spring design formulation shown in Figure 4.4(a). Figure 4.4(b) shows that the problem class corresponds to the declaration of the different types of parameters of the problem formulation. By setting requirements, the problem class is transformed into one problem instance. However, by requirements to other parameters, another problem instance is obtained.



(a) Spring design problem formulation.



(b) structure of spring design example.

Figure 4.4: *structure of spring design example.*

4.3 Complexity in Routine Design

Complexity of systems can be studied from different points of view. This thesis is based on the notions of design complexity stated in Axiomatic Design Theory (ADT) [74], explained in Chapter 2. The basic idea of this model is that without difficulty in understanding (or making, operating, etc.), a system is not complex. In this sense, complexity is the property of a system that makes it difficult to understand with the available knowledge about its constituents parts. Tomiyama further elaborates this view, by stating that complexity can be studied from the view point of knowledge structure (see Section 2.2.3), identifying two types of complexity: complexity by design and intrinsic complexity of multi-disciplinarity. The former is attributed to the structure of the design problem, while the latter deals with behavioral characteristics.

This thesis adopts ADT definition of complexity, and focuses on complexity by design. The three main reasons why this model has been chosen as framework in this research are: (1) complexity is regarded as a relative property, (2) complexity is the consequence of engineering activities, and (3) it is assumed that complexity can be managed. ADT model of complexity is used to identify different types of complexities in routine design.

4.3.1 Translating ADT Terms

In order to apply ADT theory of complexity to routine design, it is first required to relate the terms in both theories:

- ***Functional Requirements (FR's):*** correspond to the functions, performances and scenario descriptions proposed as information contents in Chapter 3. FR's model either elements or parameters.
- ***Design Parameters (DP's):*** correspond to the embodiment descriptions in the routine design formulation, and model elements and parameters.
- ***Design Matrix (DM):*** is formed by the analysis, topology and physical coherence constraints.

In order to facilitate the analysis of complexity, ADT terms are interchanged with the design formulation terms in in the following sections.

4.3.2 Model of Complexity

Figure 4.5 shows the identified complexity types in routine design. As subsequent subsections will explain, complexity of problem classes deals with incorrect problem formulations, while complexity in problem instances deals with deriving strategies for solving it. Complexity of design solutions is related to the number of elements, parameters and relations of the design solution. Physical domain

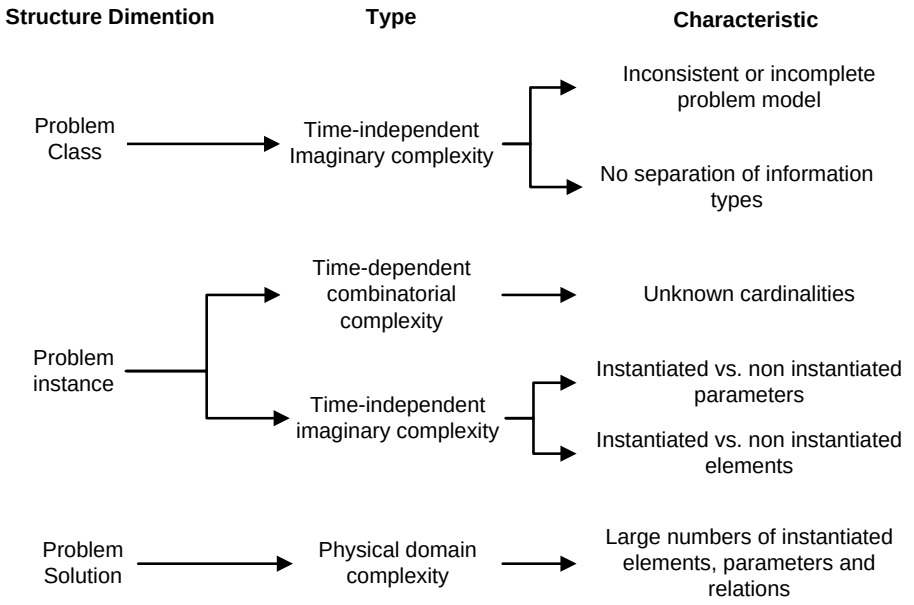


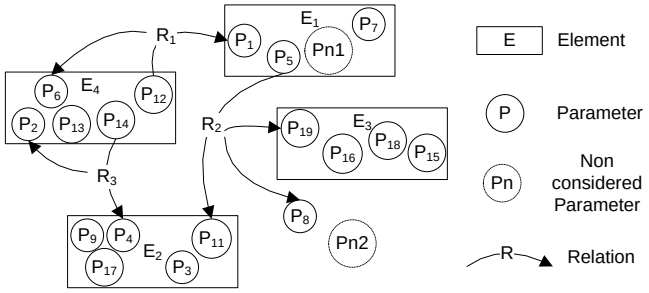
Figure 4.5: *Complexity map for routine design problems.*

complexity is out of the scope of this research, as this work focuses on solving problems rather than understanding complex solutions.

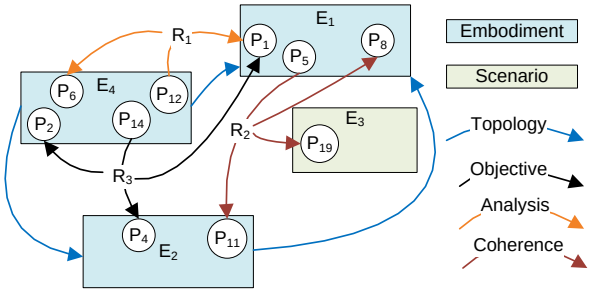
4.3.3 Complexity of Problem Classes

Time-independent complexity captures the complexity of a system in which the time dimension does not limit achieving its functional requirements. In other words, the range of the system does not change in time. In routine design, the process of moving from problem classes to problem instances is not a synthesis process by itself. This is rather a human process that involves specifying which are the parameters and elements characterizing the input of the problem. Therefore, complexity here is related to how well the problem has been formulated, and regards two types of uncertainties. One is the uncertainty of having all the required information in the problem formulation. The second is the lack of differentiation between problem chunks and its interrelations. Figure 4.6 illustrates this by sketching three possible states of a problem class according to its level of complexity. The result of both uncertainties result in incomplete and unorganized information contents. So, both aspects are caused by the lack of knowledge, or ignorance, in formulating the design problem.

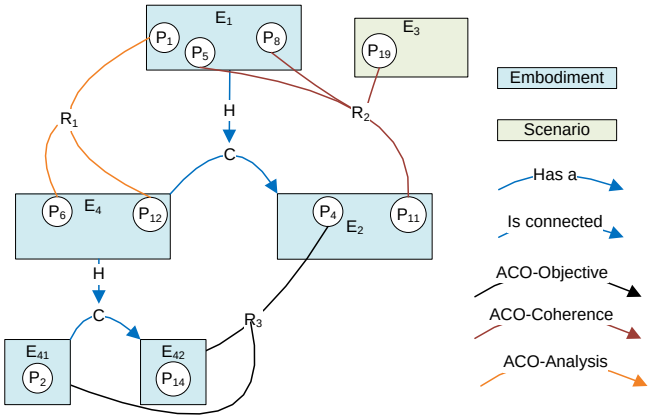
Ignorance of FRs and DPs is related to the failure to properly understand them in a design task. This is caused by a faulty or incomplete description



(a) Inconsistent and unstructured design problem.



(b) Consistent and unstructured design problem.



(c) Consistent and structured design problem.

Figure 4.6: *Three states of problem classes according to its complexity.*

of the functions, performances and scenarios in the problem formulation. As result, one would be addressing a wrong problem. Complexity emerging from incomplete problem formulations is time-independent imaginary complexity. In ADT, imaginary complexity is defined as the uncertainty that arises because of the designer's lack of knowledge and understanding of a specific design itself [74]. In order to solve this, the FRs of the problem have to be identified and related to the problem's DPs. Then, the problem can be reformulated in terms of the emerging relations between FRs and DPs. By doing so, the imaginary component of the complexity can be managed. Two methods have been developed in this work to manage time-independent imaginary complexity in problem classes:

- FBS based formulation: is presented in Chapter 5, Section 5.2. This method aids the exploration of the information required to formulate a given design problem. The goal of the method is to obtain a consistent mapping between a problem's function, its behaviors, and the parameters and relation in its formulation. The last ones following the scheme presented in Chapter 3.

- ADT based decomposition: is described in Chapter 5, Section 5.3. This method deals with the decomposition of the problem into smaller problem chunks. The goal is to distribute the information of the problem among different abstraction levels.

4.3.4 Complexity of Problem Instances

Time-independent Real Complexity

Real complexity appears in multi-objective problems, where one parameter has to satisfy contradicting objectives. This is caused when several disciplines determine an artifacts behavior. For example, the more lanes a highway has, the more traffic it can accommodate. At the same time, as the number of lanes increases, the number of accidents also increase. Designing highways with the objectives of traffic maximization and accidents minimization has a contradiction, and therefore is a problem with time-independent real complexity. This type of complexity is solved in routine design using constrain satisfaction and optimization techniques. Therefore, this type of complexity is not further treated in this work.

Time-independent Imaginary Complexity

Imaginary complexity originates from the fact that design requirements are set at different combinations of parameters and elements. As consequence, it is not known a priori in which order the problem will be solved. Furthermore, when the DM is uncoupled, a particular order is required for solving the problem. Imaginary complexity depends on the relations between known ad unknown cardinalities as well as on the relation between instantiated and non instantiated elements and

parameters. The former is regarded in this work as knowledge distribution, while the latter is regarded as requirements distribution.

From a knowledge distribution viewpoint, the more cardinalities that are known in a problem instance, the lower its uncertainty is regarding the number of elements to instantiate. Moreover, as the cardinalities of elements are often interrelated among each other, modifying one automatically leads to the modification of others. In this sense, knowledge distribution refers to the distribution of known cardinalities among the elements of the problem.

Requirements distribution, on the other hand, refers to the distribution of instantiated and non instantiated elements and parameters. For elements, complexity regards the uncertainty of having to apply bottom-up or top-down approaches, while for parameters it regards the uncertainty of knowing in which order to solve the constrained system.

Time-dependent Combinatorial Complexity

For problem instances, combinatorial complexity occurs when the design problem consists in generating complex topologies or shapes in which embodiment elements are instantiated several times within one design solution (for example, the number of gears required in a gear box). This results in a DM with time-dependent varying size and terms. When no knowledge is available about the number of instances required to satisfy the FRs, the problem presents time-dependent combinatorial complexity. One way of recognizing this type of complexity is by assessing the cardinalities of the elements in the problem formulation. When the ranges of cardinalities are not known, or cannot be written as function of other parameters, the design problems has combinatorial complexity.

Complexity Management

Complexity management of problem instances is limited to the domains of *parametric* and *topology* models, as addressed in Chapter 3, Subsection 3.4.1. The research on complexity management for the domains of *space*, *shapes* and *fields* is subject of further research.

Complexity management is performed from three different perspectives: representations, manipulating building blocks, and manipulating parameters. Chapter 6 proposes a generic model for representing design problem structures. This model is used in Chapter 7 to develop a method for translating the topology characteristics of a design problem into a algebraic model. This permits solving topologies by using algebraic solving procedures. Furthermore, a design grammars method is described for managing the combinatorial complexity of design problems. Chapter 8 presents a constraint solving algorithm for deriving solving orders of non-instantiated elements and parameters. Chapter 9 integrates these methods into a new method for CDS, which allows automating in a generic fashion problems with parametric and topology models.

4.3.5 Example: CSIM Design

According to this model of complexity, the design of CSIM has the following types of complexity:

1. Problem class, time-independent imaginary: Is caused by an inconsistent formulation. Literature does not present exact problem formulations. No separation of information contents. Furthermore, all information content is attributed to one element type channel.

2. Problem instance, time-independent imaginary: Different ways of solving DM, with different levels of difficulty. There are different approaches for placing and configuring channels.

3. Problem instance, time dependent combinatorial: Number of DPs explodes as function of time. The number of channels and its configuration is not known a priori, and changes as the design is being solved. The placement and instantiation of one channel determines the number and position of other channels.

Part III

Theories and Methods

Chapter 5

Managing Complexity I: Information Contents

This Chapter describes two methods that aim at managing complexity of problem classes. The first method focuses on how to formulate the information contents and the second on decomposing the problem into different levels of detail.

5.1 Introduction

The first challenge encountered when developing methods for CDS is the formulation of the design problem and definition of its building blocks; thus the development of the model of the artifact upon which designs can be generated. This chapter presents two methods to define a design problem's building blocks. While the first method deals with formulating "good" problems, the second deals with the distribution of the building blocks in different levels of abstraction. Both methods are paper based, and are meant to be applied previous to the development of strategies and algorithms for synthesis automation.

5.2 Method 1: FBS based Formulation

Design formulation is the active process through which a routine design problem is formally identified and described. The result is a set of representations that model the problem class. The description of the problem should explicitly reflect the structure and relations of the design problem. Therefore, the goal of FBS based formulation is to explore the set of elements, description and configuration (as defined in Chapter 3) required for formalizing a given artifactual routine design problem. The methodology, which consists of the four steps shown in Fig-

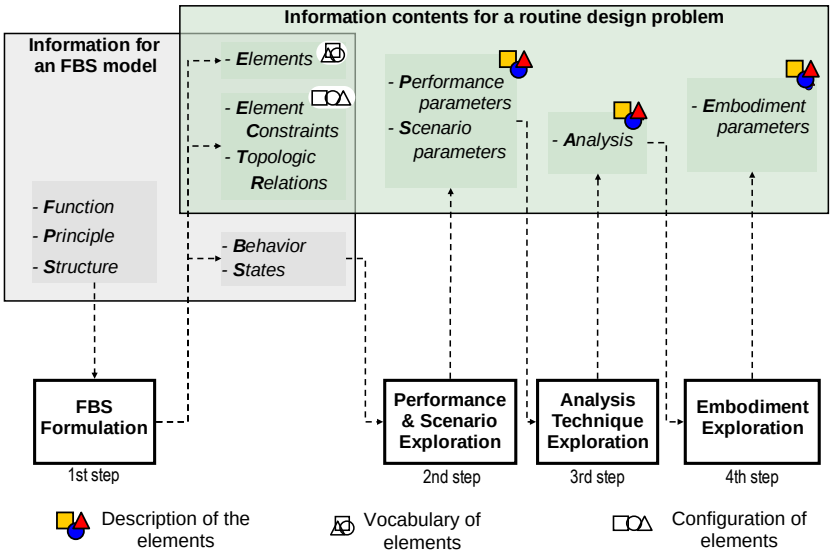


Figure 5.1: *The FBS based design formulation method.*

ure 5.1, uses FBS modeling for assessing the *function*, *behavior*, *principle*, *state* and *structure* representation of the design object in question. *States* are used for deriving performance and scenario parameters, while embodiment parameters are defined by relating performances and scenario to a given analysis technique. This method is used in M. Olthof [49] to determine the model of the design problem of a clock and the design problem of a runner gate system for injection molding. Subsection 5.2.1 demonstrated the method at the hand of CSIM design.

Step 1: FBS Formulation

The first step consists of defining the *functions*, *structure* elements and *principles* of the design problem in question, given that these are known entities when dealing with routine design. This input is used to formalize the *behaviors* that support the *functions*. The (sequence of) *states* that the *structure* is meant to experience are described by relating the design *structure* elements and the derived *behaviors* to *principles*. Topology relations can be identified by formalizing the connectedness of the elements in the structure.

Step 2: Performance and scenario exploration

Parameters describing the properties of *states* or sequence of *states* can now be derived from the FBS formulation. For this purpose, the parameters present in

both *principle* and *state* serve as entities representing performance and scenario parameters. The dynamics implied by the sequence of *states* is also translated into a performance parameter.

Step 3: Analysis Technique Identification

The analysis technique can now be identified by assessing the *principles* ruling the behavior of the design artifact. This is done by assessing existent techniques (e.g. literature, CAE, expert knowledge) or by deriving a new method. In either case, the analysis method has to integrate the performance and scenario parameters identified in the third step of the method. Furthermore, depending on the desired level of detail -or abstraction-, different analysis techniques can be used.

Step 4: Embodiment Definition

The analysis technique can now be used to define the parameters describing the embodiment for the chosen abstraction level. This is done by identifying the set of parameters present in the analysis technique describing the characteristics of the artifacts structure.

Step 5: Design Rules and constraints

Once the performance, embodiment and scenario parameters have been identified, the constraints and design rules can be formalized. Element constraints can now be represented mathematically by relating the found parameters. Confinement constraints are stated by defining the ranges of values of each embodiment parameter. Confinement constraints are meant to avoid physical impossibilities by confining the parameters between limits the designers considers appropriate, depending often on experience. To specify design rules, knowledge has to be assessed. Having the parametric descriptions and constraints can aid the process of extracting knowledge from diverse sources, like design books, experts and Internet. In appendix a method for eliciting expert knowledge is presented.

5.2.1 Example: CSIM Design

Figure 5.2 shows an overview of the information flow of this method for CSIM design. The following subsections further detail its application.

Step 1: FBS Formulation

Consider the design of cooling systems for injection molding shown in Figure 5.3. Here, the cooling system *functions* are mapped to *behaviors* that allow the required *function* to exist. For example, the *function* 'Cool Down Plastic Part' is mapped to the *behavior* 'Extract Heat From Plastic Part'. The figure also shows how *structure* and *behavior* are related to the *states* by means of a *principle*, in

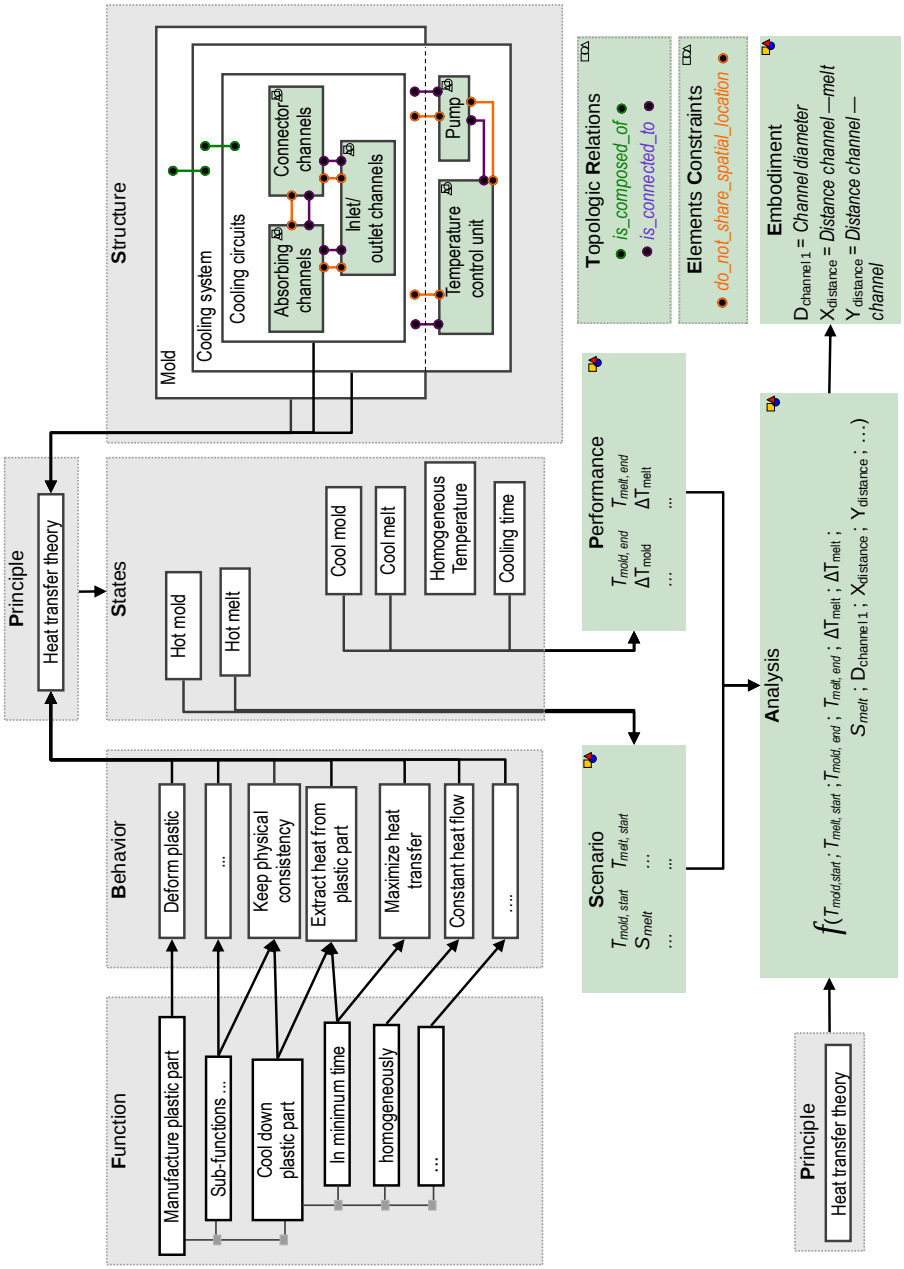


Figure 5.2: *Overview of the design exploration method in the design of cooling systems for injection molding.*

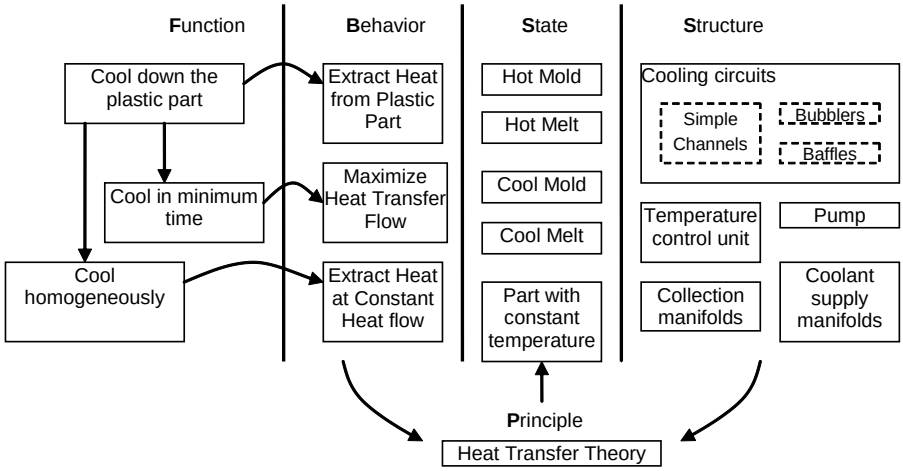


Figure 5.3: *FBS description of CSIM design.*

this case heat transfer theory. The *structure* is composed of several components. However, for explanation purposes, only cooling channel layout design is taken into consideration. Examples of topology relations are:

- cooling circuit *is_composed_of* cooling channels
- one cooling channel *is_connected_to* one cooling channel

Examples of physical coherence constraints are:

- cooling channel *does_not_share_spatial_location_with* melt.
- cooling channel *starts_at* mold surface
- cooling channel *ends_at* mold surface
- cooling channel *is_confined_in* mold

Step 2: Performance and Scenario Exploration

Table 5.1 presents a generalized summary of performance and scenario parameters. The dynamics implied by the sequence of *states* is also translated into a performance parameter. In the case of cooling system design, the difference in time between both states is one performance parameter that describes the time required to change from one state (hot melt) to the other(cold melt). The complete description of performance and scenario parameters is presented in Table 5.1.

Table 5.1: *State based performance and scenario mapping.*

State	***Sequence***	***State variables***	***Type***
Hot Mold	First	Temperature Mold	Scenario
Hot Melt	First	Temperature Melt	Scenario
Cool Mold	Second	Temperature Mold	Performance
Cool Melt	Second	Temperature Melt	Performance
Part constant Temperature	Second	Temperature Difference in Part	Performance

Step 3: Analysis Technique Identification

Consider the case of the cooling system design for injection molding. A simple analysis has been chosen from [62]. The analysis is derived from the equation of heat transfer by conduction for a substance in rest under several assumptions. The analysis consists of the following calculation:

1. Calculate heat transfer from the melt to the cooling medium using equation 5.1.

2. Calculate the shape factor (S_e), Reynolds (Re) number, the coolant heat transfer coefficient (α) and the thermal diffusivity of the melt (a) using equations 5.2, 5.3, 5.4 and 5.5 respectively.

3. Consider $Q_{abs} = Q_m$, and use results of 5.2, 5.3, 5.4 and 5.5 to solve the system composed by equations 5.6 and 5.7.

The set of equations 5.1 to 5.7 represent the analysis technique that has been chosen for the purpose of demonstrating the methodology. Each one follows from different partitions, obtaining that different parameters appear in different analysis equations. The analysis has been formulated in such a way that it holds at both sides of the axes of symmetry shown in Figure 5.4. This allows assessing the temperature difference between the two sides of the mold, which was selected as performance parameters in the second step of the method. It is worth mentioning that 3D analysis formulation is better suited for the performance calculation of this type of problems. However, to describe the performance and scenario parameters so far considered, the analysis technique here presented suffices. Table 5.2 presents a list with the description of the used symbols.

$$Q_{abs} = 10^{-3} \cdot \left[\left(T_M - T_E \right) C_{ps} + i_m \right] \cdot \varsigma_m \cdot \frac{s}{2} \cdot x \tag{5.1}$$

$$S_e = \frac{2\pi}{x^4} \cdot \left[\ln \left(2 \cdot x \cdot \sinh \left(\frac{2 \cdot \pi \cdot y}{x} \right) \Big/ \pi \cdot d \right) \right]^{-1} \tag{5.2}$$

$$Re = \frac{10^{-3} \cdot u \cdot d}{v} \tag{5.3}$$

$$\alpha = \frac{0.031395}{d} \cdot Re^{0.8} \tag{5.4}$$

$$a = \lambda / \varsigma \cdot C_p \tag{5.5}$$

$$t_k = \frac{10^{-3}}{Q_m} \cdot \left(\frac{1}{\lambda_{st} \cdot S_e} + \frac{1}{\alpha \cdot 10^{-3} \cdot 2 \cdot \pi \cdot R} \right)^{-1} * (T_W - T_{water}) \tag{5.6}$$

$$t_k = \frac{s^2}{\pi^2 \cdot a} \cdot \ln \left(\frac{4}{\pi} \cdot \frac{T_M - T_W}{T_E - T_W} \right) \tag{5.7}$$

Step 4: Embodiment Definition

Embodiment parameters are now identified by selecting the structural variables present in the analysis relations. For the case of design of cooling systems, the embodiment parameters are sketched in Figure 5.4. In Appendix B, a complete description of the elements, relations, parameters and constraints of CSIM are presented.

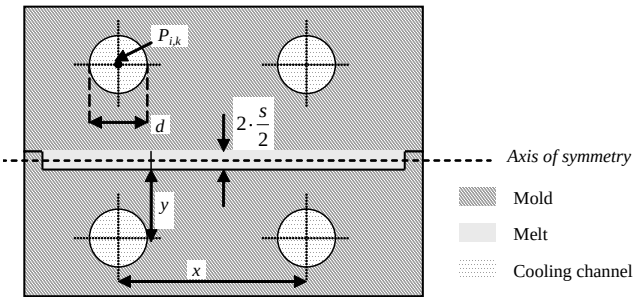


Figure 5.4: *Embodiment definition in FBS based formulation method.*

Design Rules and Constraints

For the case of cooling channels layout design, some examples are:

1. The topologic relation, cooling channel *is_confined_in* mold, can be modeled by equation 5.8

Table 5.2: *Design parameters in CSIM*

Symbol	**Units**	***Description***	***Classification***
Q_{abs}	KJ/m	Heat content of melt	Scenario
Q_m	KJ/m	Heat absorbed by coolant	Performance
t_k	s	Cooling time	Performance
S	mm	Part thickness	Scenario
X	mm	Distance X	Embodiment
Y	mm	Distance Y	Embodiment
D	mm	Diameter of cooling channel	Embodiment
R	mm	Radius of cooling channel	Embodiment
$P_{i,k}$	(mm, mm)	Point i of cooling channel k	Embodiment
i	No	Index: 1 is start and 2 is end point of cooling channel	Embodiment
k	No	Index: 1... N cooling channel	Embodiment
T_M	^0C	Melt (Molding) temperature	Scenario
T_E	^0C	De-molding temperature	Performance
T_W	^0C	Temperature of mold	Performance
T_{water}	^0C	Temperature of cooling water	Scenario
i_m	KJ/Kg	Latent heat of fusion of the polymer	Scenario
C_{ps}	KJ/KgK	Specific heat of the polymer	Scenario
ς	g/cm^3	Melt density	Scenario
a	cm^2/s	Thermal diffusivity of the melt	Scenario
V	m^2/s	Kinematics viscosity of water	Scenario
U	m/s	Velocity of cooling water	Scenario
λ_{st}	W/mK	Thermal conductivity of mold steel	Scenario
C_p	$KJ/KgK)$	Specific heat of water	Scenario
λ	W/mK	Thermal conductivity of water	Scenario
α	W/m^2	Heat transfer coefficient	Scenario
t_k	s	Cooling time	Performance

2. Element constraint avoiding two cooling channels sharing the same space in equation 5.9
3. Design rule defining the diameter of the cooling channel as function of the part thickness in equation 5.10

$$P_{i,k} \in Mold \tag{5.8}$$

$$\sqrt{(x_{i,k} - x_{i,k+1})^2 + (y_{i,k} - y_{i,k+1})^2} - \frac{(D_K + D_{k+1})}{2} \leq \zeta_1 \tag{5.9}$$

$$\zeta_2 \leq D \leq \zeta_3 \tag{5.10}$$

Note that the symbol ζ_i is used for representing a border value, while the word *Mold* represents the element mold.

5.3 Method 2: ADT based Decomposition

The goal of this method is to decompose the design problem class into several independent levels of abstraction. Decomposing the design into smaller chunks -a divide and conquer strategy- is often used to reduce complexity [77], as for example in ADT [73] and MDO [1]. However, the resulting chunks are required to have clear boundaries and governing principles. According to Goel et al. [21], design decomposition has to be based on problem structure. This avoids unwanted side effects, such as multiple interdependencies among sub-problems and disjointed sub-problems. Therefore, the method presented decomposes the problem at the hand of the problem structure. The method (applied both in the functional domain and in the physical domain) consists of the three steps shown in Figure 5.5:

- Identification: determines the space of FRs and classifies the types of DPs involved in the problem.
- Reformulation: assembles the obtained FRs and DPs into a new decomposed problem model
- Separation: separates the reformulated problem in abstraction groups according to the problem's DM.

In Subsection 5.3.3 the method is applied to the design CSIM. In Jauregui Becker et al. [29] the method is applied to the design of Printed Circuit Boards.

5.3.1 Functional Domain

In the functional domain, the method consists of identifying functional elements, identifying its instantiation order and encapsulating the resulting structures into new problem formulations. Functional elements are considered high level DP related to one (or a group of) FR(s).

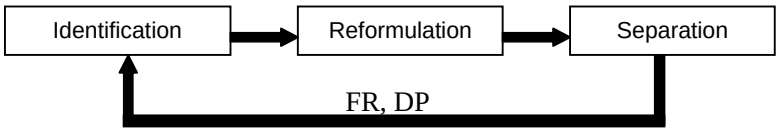


Figure 5.5: *The ADT based decomposition method.*

Functional Decomposition

An artifact overall function is usually composed of a set sub-functions attributed to individual or groups of elements. Functional decomposition methods have been broadly used to assist conceptual design phases, as for example in ICDM (Integrated, customer driven, Conceptual Design Method) [23]. In this research, identification in the functional domain allows obtaining the functional map of the artifact being designed. This is achieved by:

1. Listing the functions of each of the elements involved in the formulation.
2. Elements undergoing more than one non-additive function (independent functions) are split into new element definitions: one element for each function.
3. Formalizing how one element's function acts upon other elements.

Reformulation

The reformulation step consists in making a new problem formulation based on the functional elements that resulted from applying the previous step. To do so, the new topology relations among functional elements are formalized. In addition, the topology relations of the mother elements are inherited by the new elements. The direction of the emerging topologic relations among functional elements equals the direction in which functions are applied among them. This follows from the principle that the direction in which functions are applied expresses how one element instantiation is limited by the previous instantiation of others. The resulting map of elements and topologic relations can be seen as the set of syntactic rules indicating the constraints imposed in the elements instantiation order.

Separation

Now that all emerging relations have been formalized, it is assessed whether the design problem can be separated into smaller chunks. According to ADT, this is possible if the DM is uncoupled or decoupled. Therefore, the separation step consists in assembling the DM of the reformulated problem and deriving an appropriate instantiation order that manages the time-independent imaginary complexity of the system.

5.3.2 Physical Domain

The application of the method in the physical domain is based on the design descriptions classification presented in 3.4.1. This classification is used to decompose elements in different primitives, where each primitive encapsulates attributes that correspond to one type of description. By doing so, building blocks with focused search ranges are found. Furthermore, the method manages imaginary complexity by firstly integrating physical coherence constraints into the resulting primitive elements, and secondly by introducing periodicity in the system.

Identification

First, embodiment and scenario parameters are classified in one of the following groups:

- Parameter: models the properties applicable to a whole element. These can be of different nature, as for example numeric, symbolic, logic, predicate and combinations among them.
- Space: describes the position of the elements and depends on the chosen coordinate system (Cartesian, Cylindrical of Spherical) and the dimensions of interest (1D, 2D, or 3D).
- Field: uses parameters and geometric vectors to describe properties that hold for specific regions of the elements. Fields are specified together with an incident zone, which is the spatial place where the field influences an element. An incident zone can be a volume, an area, a line or a point.
- Shape: describes the form of an element or groups of elements. Commonly used models are based on geometry and shape graphs.
- Topology: uses topology relations to describe the disposition of elements in design. Cardinality is used to measure the number of times an element is instantiated in the topology.

For example, the layout design of boxes would have spatial descriptions (its position) and shape descriptions (the length, width and depth of the boxes).

Reformulation

Reformulation consists of encapsulating each of the identified description dimensions into a new primitive element and formalizing the physical coherence constraint among them. In this way, a new problem formulation is obtained. In the previous example this would mean that a box is composed out of a primitive element shape and a primitive element position.

Separation

This step consists of separating the reformulated problem into smaller chunks by assessing the DM of the resulting problem. As the interest here lays in the physical domain, the DM has to describe the interrelations among the primitive elements, or DPs, of the problem. As result, primitive elements are set to different levels of abstraction.

5.3.3 Example: CSIM Design

The example here presented is based on the model present in Figure 5.6, which is a simplification of the results gotten in the previous example.

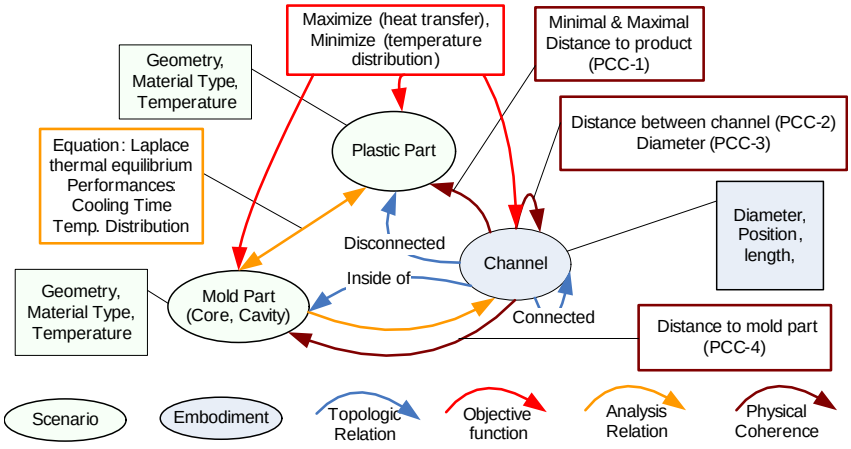


Figure 5.6: *Problem formulation of CSIM design.*

Functional Domain: Decomposition

Consider the case of injection molding cooling systems. To decompose the element Channels we start by listing its functions:

- *Function 1:* to cool down the melt. Channels are placed close to the part geometry and arranged such that heat is transferred in a homogeneous manner from the melt to the coolant flowing through it.
- *Function 2:* to transport the coolant. Coolant is transported between channels absorbing heat (function 1) to constitute cooling circuits.
- *Function 3:* to exchange coolant with the environment. Channels are used to connect the cooling circuits with the external surface of the mold.

As these functions are not additive but complementary, the element channel is decomposed into three different elements: (a) Absorber channels to fulfill function 1; (b) Exchanger channels to fulfill function 2; (c) Connector channels to fulfill function 3. Figure 5.7(a) presents the resulting channels and their functions assembled in a model. As shown, the new element Absorber channel applies its function to the scenario element Plastic Part, Connector channels apply their function to the element Absorber channel, and the Exchanger channel applies its function to both the Connector channel and the Absorber channel.

Functional Domain: Reformulation

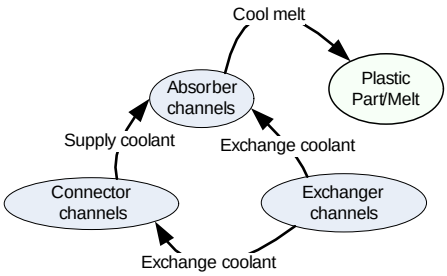
In Figure 5.7(b) the reformulated CSIM is presented. The figure shows both the functional elements and the topology relations that have emerged. Here, each channel element type can be connected to another channel element of the same type (relations 3, 5 and 9). Connector channel elements can be connected to Absorber channels (relation 12). However, the relation does not work in the opposite direction. This is because Connector channels function is constrained by the previous existence of an arrangement of Absorber channels, as indicated in Figure 5.7. A similar situation occurs for the relation between Exchanger channels and the partial circuits formed by the arrangement of Absorber and Connector channels (relations 6 and 8): the existence of the former is constrained by the previous existence of the latter.

Functional Domain: Separation

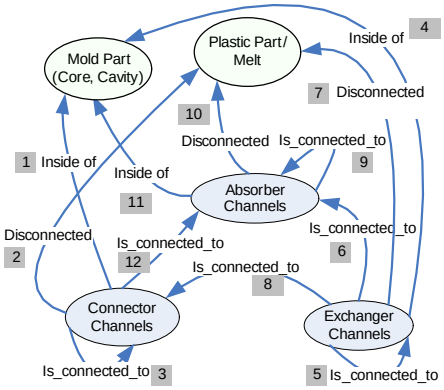
For the case of CSIM, a DM is assembled by considering the relations between the FRs and the DPs shown in Figure 5.7, obtaining:

$$\begin{Bmatrix} FRs \\ Cool\ Melt \\ Transport\ Coolant \\ Exchange\ Coolant \end{Bmatrix} = \begin{bmatrix} DM \\ 9 & - & - \\ 12 & 5 & - \\ 6 & 8 & 3 \end{bmatrix} \begin{Bmatrix} DPs \\ Absorber\ Channel \\ Connector\ Channel \\ Exchanger\ Channel \end{Bmatrix} \quad (5)$$

As shown, the DM is decoupled and square, and can be separated into three independent problem formulations. The first is shown in Figure 5.8(a). Here, Absorber channels are first instantiated by taking into considerations the relations shown in the figure. In a similar manner, Connector channel are designed considering the Absorber channel as scenario elements, shown in Figure 5.8(b). The formulation in Figure 5.8(c) is assembled for the design of Exchanger channels. If the number of FRs does not match the number of DPs, the resulting DM is not square. In such a case, a DM of the relation among the DPs is better suited for determining the appropriate instantiation orders.



(a) Functional elements in cooling layout design.



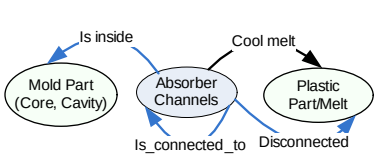
(b) Decomposed cooling layout design.

Figure 5.7: *Reformulation of CSIM.*

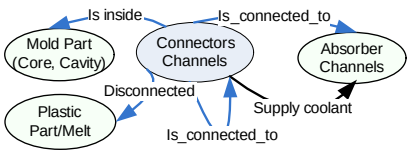
Physical Domain: Identification

In this step, descriptions are categorized around the five attributive dimensions. In the case study of CSIM design, the identification of primitive elements is done by analyzing the type of description used to model them. As the elements Absorber channels, Connector channels and Exchanger channels are modeled with the same descriptions, the identification has been generalized in an element Channel, which represents either one of the three before mentioned types. Channels have attributes in two domains:

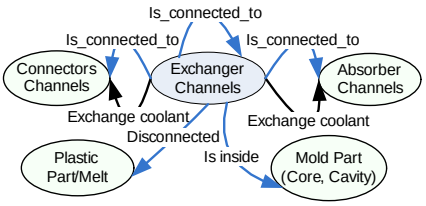
- Space: models the position of a channel inside the mold. The position is modeled by three coordinates in the Cartesian coordinate system.
- Shape: models the geometry of a channel by the descriptions diameter (D) and length (L).



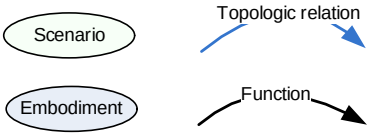
(a) Sub-problem 1: absorber channel design.



(b) Sub-problem 2: connector channel design.



(c) Sub-problem 3: Exchanger channel design.



(d) Legend

Figure 5.8: *Resulting decomposed problem formulations.*

The scenario element Mold Parts can be modeled in different dimensions: Shape, Fields and Parameter. As Mold Parts are not subject of design and its shape and parametric representation are fixed, it is chosen to use the representational dimension Fields.

Physical Domain: Reformulation

For the case of CSIM shown in Figure 5.9, the following primitive elements are derived:

- Points: encapsulate the Space dimension of the representations. A Point contains a description about its position (x,y,z). Points are related to the scenario elements by the physical coherence constraints PCC-1 and PCC-4. A set of Points indicates the path followed by the channel.
- Segments: encapsulate the Shape dimension. A Segment contains the descriptions diameter (D) and length (L). The physical coherence relation PCC-2 determines the minimum allowed distance between two points to avoid break of the mold, while the relation PCC-3 determines the diameter of the channel as function of the distance between a Point and Mold Part or Plastic Part.

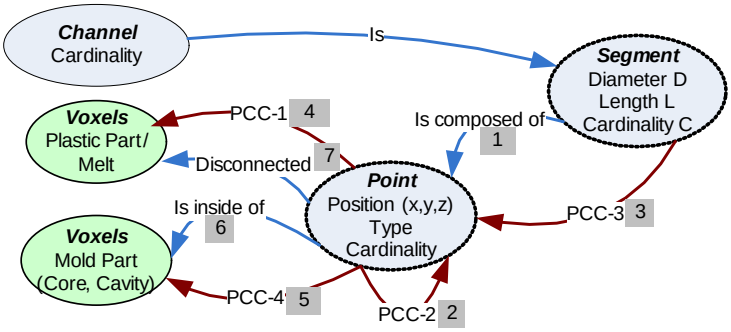


Figure 5.9: *Primitives in functional element "Absorber Channel".*

Given that the three functional elements are decomposed into the same primitives, the following notation is used to differentiate among them:

- Absorber channels have primitives Blue Points and Blue segments.
- Connector channels have primitives Green Points and Green segments.
- Exchanger channels have primitives Brown Points and Brown segments.

These six primitive elements define the information contents of the DPs of the problem. On the other hand, the element Mold Parts is also decomposed into primitives. Here, the dimension of concern is Fields and, as such, voxel elements are chosen. Voxels are cubic units and can be used to describe a mold's shape and position.

Physical Domain: Separation

For the case of CSIM design, the resulting DM becomes:

$$\begin{Bmatrix} \text{Points} \\ Segments \\ MoldPart \\ PlasticPart \end{Bmatrix} = \begin{bmatrix} 2 & 0 & 5,6 & 4,7 \\ 3,1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{Bmatrix} Points \\ Segments \\ MoldPart \\ PlasticPart \end{Bmatrix} \quad (6)$$

According to this, Points have to be instantiated firstly and Segments secondly. Now that the problem has been separated, the combinatorial aspect of the complexity is tackled. In the case study of CSIM, combinatorial complexity arises from the uncertainty of how many Channels are required and how they should be connected. This is managed by predefining a 3D grid of points without "color" within the voxel mesh. By doing so, periodicity is brought in the system as suggested in ADT. By setting the distance between Points according to relation

Table 5.3: *Logic relations determining the color of Points.*

Point	***ID***	***Logic relation with scenario elements***
Blue	C1	Surrounded by [(core voxels) OR (cavity voxels)] AND [product voxels]
Green	C2	Surrounded by [(core voxels) OR (cavity voxels)]
Brown	C3	Surrounded by [(core voxels) OR (cavity voxels)] AND [exchanger voxels]
Black	C4	Surrounded by [non drillable voxels]

PCC-2, imaginary complexity is also removed from the system, as relation 2 can be removed from the DM shown in equation 6. In order to integrate the physical coherence constraints PCC-1 and PCC-4 into DPs, one extra type of Points is declared, namely, Black Points. Black points define the positions where channels cannot be placed to avoid mold breakage. In Table 5.3, the logic relations that determine the colors of Points are presented.

Results

Figure 5.10 shows the results of applying this method. A hierarchical model consisting of three levels of abstraction (Points design, Segment design and Channel design) is obtained. Each element in the model is considered a DP, while the topology relations among them represent the syntactic rules determining their structure. Physical coherence constraints represent semantic rules assuring no physical inconsistencies occur. The complexity of the system has been reduced by specifying the ranges of each DP: Points as function of the scenario elements Plastic Part and Mold Parts, Segments as function of Points, and Channels as function of Segments. Furthermore, the order of instantiation was determined for each abstraction level at the hand of its DM.

The representations in Figure 5.10 can now be used for automating the design of CSIM. Chapter 9 describes the implementation of this method to automate CSIM design using these representations.

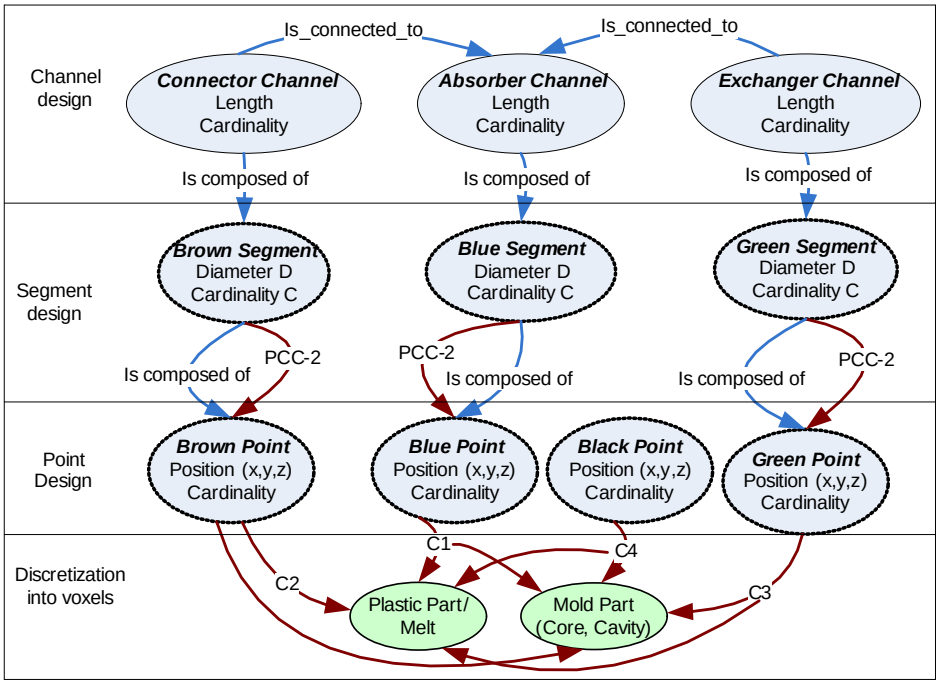


Figure 5.10: *Results of decomposing the CSIM design problem.*

Chapter 6

Managing Complexity II: Representations

This Chapter presents a framework for representing artifactual routine design problems according to the structure described in Chapter 4. First, the four basic building blocks (Elements, ACO-relations, C-relations and H-relations) are proposed. Later, an example describing how to translate a routine design problem into this framework is presented.

6.1 Introduction

A great diversity of representation schemes can be found in CDS literature. While some representations are specific for one domain (e.g. super quadric for shape design), other representation schemes, like grammars and parametric models, have proven to be of a more generic character. For example, design grammars have been successfully used in computational synthesis methods for shapes design (e.g. [42, 51]), product design configurations (e.g. [71]) and conceptual design (e.g. [34]). However, these representations are specific to the problem formulation model and its automation approach. In order to be capable of developing methodologies for complexity management, standard representation are required. The building blocks should be capable of representing different levels of detail independent from each others, and supporting the instantiation of elements, parameters and relations independent from hierarchical dependencies.

Considering these requirements, a framework to represent the structure of artifactual routine design problems has been developed. The framework is named Topology Abstraction Representation Diagram (TARD), as the focus is set on the topology characteristics of design problems. As this thesis concerns computational design synthesis, software implementation aspects have been considered. The

framework is founded upon two existing representation models, namely, multi-level networks and graph grammars. This section describes the generalities of these models as well as its shortcomings for representing different types of design problems.

6.1.1 Multi-level Networks

Nested topologies require an approach for separating different levels of design detail. Johnson et al. [31] does so by presenting a multilevel network for relating elements in a topology. This network distinguishes parts and its components at different levels of abstraction. Components at level N are related by an assembly relation R to one component at level $N+1$. By doing so, the component on the higher level $N+1$ can be represented as function of the components on the lower level N and the assembly relation R, as shown in the example in Figure 6.1. Here, an arch is represented as function of the blocks $b1$, $b2$, $b3$ and the relation R. This makes the components of the multilevel network highly dependent across levels of abstraction, thus in a vertical direction. As a consequence, the components within a level are not directly related with each other in a horizontal manner. This multilevel network only uses an implicit assembly relation between the levels of detail. In order to obtain an explicit relation between detail levels, the assembly has to be managed autonomously in the lower level, requiring another method for the establishment of topological relations.

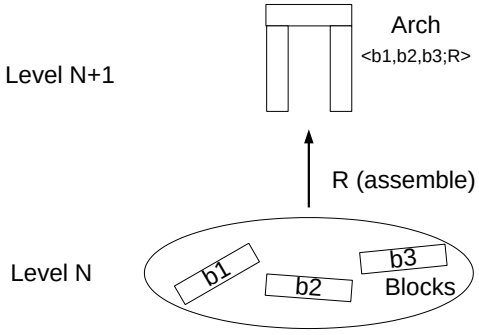


Figure 6.1: *Vertical assembly relation in a multilevel network.*

6.1.2 Graph Grammars

As described in Chapter 2, grammars make use of rules to produce networks of elements by either modifying existing graphs or expanding an initial single element graph. A grammar rule, as shown in Figure 6.2, specifies how one sub-graph can be replaced by another sub-graph. By applying grammar rules, different components

can be inserted to form a design topology. The example in Figure 6.2 illustrates a simple transmission system. As shown, rule $R1$ describes the connection of element $inputshaft$ and $gear1$, rule $R2$ describes the connection of element $gear1$ with element $gear2$ and rule $R3$ describe the connection of the lement $gear2$ with the element $outputshaft$. Grammars make use of "*horizontal*" relations in opposition to the multilevel network representation, which uses a "*vertical*" relation. In other words, grammars describe the relation of elements at one level of detail, while multilevel networks describe the relation between elements at different levels of detail. For the case of example in Figure 6.2, the transmission system is represented explicitly by the components and their connectivity relations and not by a single implicit assembly relation like in the multilevel network.

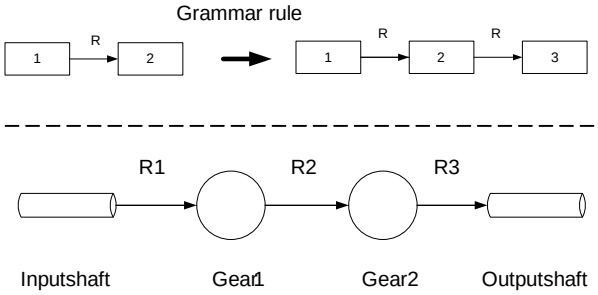


Figure 6.2: *Example of horizontal grammar representation.*

6.2 Theory 1: TARD Model

The Topology Abstraction Representation Diagram, or TARD, integrates explicit relations to model both the connectedness among elements in one abstraction level and the inter-dependency of elements at different abstraction levels. By doing so, characteristics of grammar, parametric and multi-level networks representations are resembled by a small set of generic building blocks: Elements, C-relations, H-relations and ACO-Relations. Elements represent individual components of the design problem, and group the set of parameters used in its description. For example, in Figure 6.3 the components *E1*, *E2* and *E3* represent an engine, a gear box and a wheel respectively. C-relations represent the connectedness of the elements in the topology, which is described in Figure 6.3 by the relations C. H-relations model how a group of C-relations are related to describe the composition of a higher level element. This group of C-relations, together with their corresponding Elements is denoted here as *abstraction-group*. ACO-relations are used to model analysis relations, physical coherence constraints and objective functions.

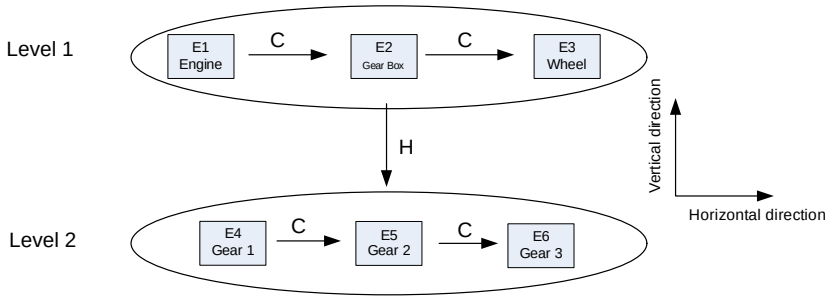


Figure 6.3: *Example for the general usage of Elements, C-relation and H-relations on two levels of detail.*

Using these building blocks to represent a problem structure results in a horizontal and vertical connected network, as shown in Figure 6.3. By considering explicit -instead of implicit- relations, TARD supports the creation of problem solutions at one level of detail topologically independent from other levels of detail. The last is required for supporting bottom-up and top-down design strategies simultaneously. TARD reassembles the design structure presented in Chapter 4, as shown in Figure 6.4. In this sense, the building blocks in TARD have both a class representation and an instance representation. The problem class is constructed at the hand of Elements, C-relations, H-relations and ACO-relations classes. A problem instance is defined by partiality instantiating these building blocks.

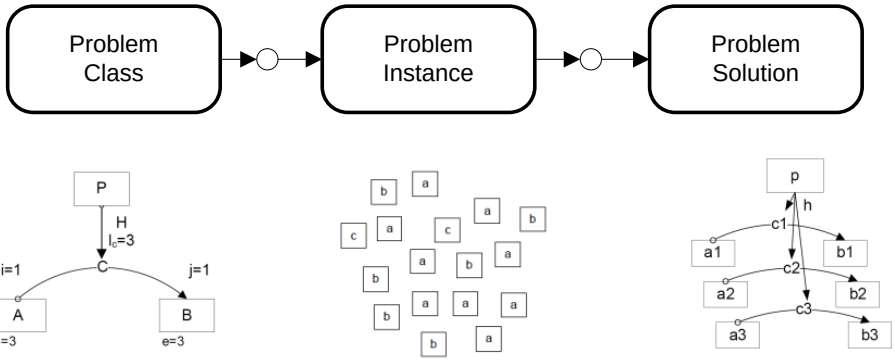


Figure 6.4: *Design problem structure using TARD.*

6.2.1 Base definitions

Cardinality

In order to control the number of instances of each building block, TARD uses cardinalities. Two types of cardinalities are distinguished: local and global. While local cardinalities describe the number of instances in a relative fashion (e.g. C-relation in relation to an Element), global cardinalities model the total number of instances of a certain element class in the complete design problem.

References

A reference is a pointer to another class or object instance. It is used to define how one building block is related to another. Relations also use references to element parameters to define its form. Depending on the direction of the association, the reference is of the type *owned* or the type *owner*. Owner references follow the hierarchical structure from top to bottom, while owned references do it from bottom to top. References are required to enable communication among the building blocks classes and instances.

6.2.2 Building Blocks

Elements, C-relations and H-relations are used at the three levels of problem structure: problem class, problem instance and problem solution. At the problem class level, these building blocks describe the blueprint of an artifact. At a problem instance level, a combination of class and instance descriptions is used, while at the problem solution, only instances of these building blocks are used. From this perspective, the problem instance can be seen as a partially instantiated problem solution. The following subsections present a detailed description of each building block of TARD.

Elements

An Element represents a class description of a component in a design problem. For example, the transmission system in Figure 6.3 contains four Element blocks: *inputshaft*, *gear1*, *gear2* and *outputshaft*. Each Element class is described in TARD by the following information:

- **Parameters**: Describe the class by modeling its characteristics. At the problem class, parameters are only declared. Values are attributed either by setting the requirements at the problem instance level, or by a generation algorithm to create problem solutions.
- **Cardinalities**: The total cardinality e describes the amount of Elements of a type in the design solution, while the local cardinality e_l describes the total amount of Elements within one abstraction-group. Cardinality can be

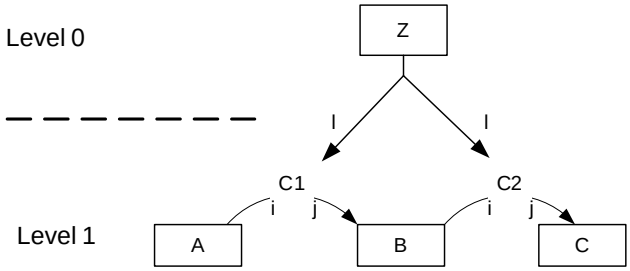


Figure 6.5: *Example of bi-level TARD Diagram.*

either known or unknown. Depending on whether cardinalities are known or unknown, the problem exhibits different complexities, as described in Chapter 5.

- **References**: owner-reference to a H-relations and owned-reference to a C-relation.

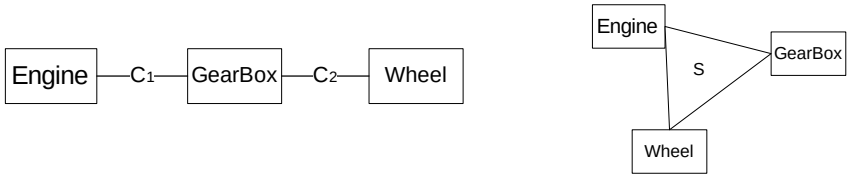
Elements can be classified according to their absolute position or relative position in the topology diagram. From an absolute perspective, three types of Elements are distinguished:

- **Zero Elements**: Each network formulated by the representation of this framework has a hierarchical structure topped by a single Element representing the structure as a whole. In Figure 6.5, the zero level Element is denoted as Element Z. The cardinality of Zero Elements equals one (1) and no C-relations are connected to them.

- **Abstract Elements**: Are those with a subordinate abstraction-group, and are found between Zero Elements and Base Elements. In Figure 6.5, Z is both a Zero-element and an abstract element.

- **Base Elements**: Are those at the bottom of each network path, and do not have lower abstraction level dependencies. In Figure 6.5, elements A,B and C are all base elements, as none of them has a lower abstraction-group.

On the other hand, from a relative perspective elements can be classified in parent or son elements. For example, in Figure 6.5 the element Z is the parent element of A, B and C; while element B is the son of element Z.



(a) 3 Elements connected by two 1-simplices.

(b) 3 Elements connected by one 2-simplex.

Figure 6.6: *Connectivity relations.*

C-Relations

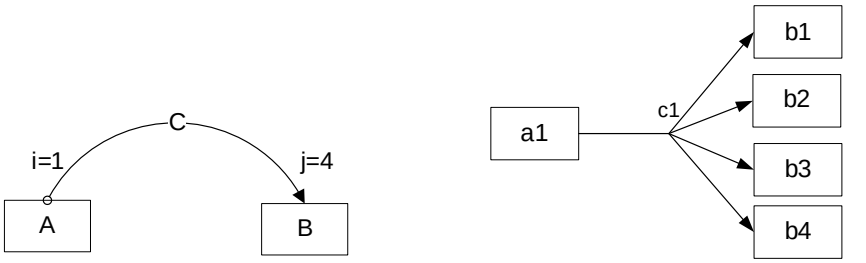
C-relations are used to determine the connection between two element classes in an abstraction-group. Having explicit connective relations makes it possible to assemble sequences of Elements independent of the previous existence of instantiated parent elements. Elements are horizontally related to C-relations by references, which allows its instantiation in two ways: by instantiating the relations and later their referenced Elements (a *top-down* strategy), or by matching previously instantiated Elements and then instantiate the C-relation accordingly (a *bottom-up strategy*). Consider the example shown in Figure 6.3. Here, the Element engine can exist without the need for an Element wheel or even without the need for a C-relation C.

Furthermore, C-relations support the generation of topologies by generating sequences, enabling the application of grammar approaches for design synthesis. In contrast to the multidimensional representation, the grammar approach with the rule in Figure 6.6(a) relates two Element classes with each other using a 1-simplex to describe the relation. By an iterative repetition of the rule, a multitude of C-relations can be created. TARD only supports the representation of 1-simplex C-relations, thus, it only relates two elements. Higher dimensional simplices are indirectly supported by the H-relation, as it it described in Subsection 6.2.2. C-relations are modeled using the following descriptions:

- **Direction**: Indicates how one Element is related to other Elements of another class.
- **Cardinalities**: C-relations have two local cardinalities, namely, cardinality i and cardinality j. These cardinalities define the number i of instances of an Element type that can be connected with a number j of instances to another element. As the framework is decided to suport 1-simplex relations, i has a constant value of 1. The example shown in Figure 6.7(a) illustrates a C-relation class C that relates the Element classes A and B with $i = 1$ and $j = 4$. The instanced C-relations in Figure 6.7(b) shows that one instance of A is connected to four instances of B.

- **References**: Owner-reference to an element, owned-reference to a H-relation
- **Parametric models**: Are rules concerning the constitution of the connected Element instances. These rules, in the form of equation, relate parameters of the two Element classes involved in the relation. Figure 6.8 shows an example, where the parameters defined within the two Element classes are related to each other by a set of rules in the C-relation. Accordingly, the instantiation process of Element 2 has to consider the parametric relations defined with respect to the prior Element.

Note: It is important to notice that although C connects one Element A with four Elements B, it does not connect the instances of B ($b1$,$b2$,$b3$,$b4$) among each others. So, even if C-relations relate more than two Element instances, as in the example, they do not model multidimensional simpleces.



(a) C-relation class connected to two Element classes.

(b) C-relation instance connected to five Element instances.

Figure 6.7: *C-relations: class and instance.*

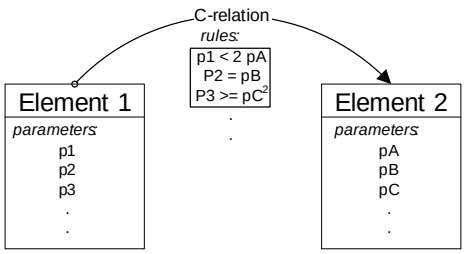


Figure 6.8: *Parametric rules in the C-relation relate the parameters of the connected Elements.*

H-relations

H-relations connect the model in a vertical direction, thus they relate two levels of detail with each other. More precisely, a H-relation connects an abstract Element in a level N with an abstraction-group on a more detailed level $N + 1$. *Abstraction-groups* describe topology compositions on an abstract Element, and are analoguos to the assembly relation of the multilayer network [28]. However, while assembly relations in the multilayer network relates a set of components to a single component on a higher level, the H-relation relates the C-relations of abstraction-group at level N with its parental Element on level $N - 1$.

A topological diagram according to TARD is composed of elements connected by relations. These relations represent the possible topological dependencies of a design problem. By referencing H-relation to C-relations instead of directly to Elements, the combination of the H-relations and C-relations establishes a self contained definition of the structure of each abstraction-group. By doing so, Elements and topology relations can be uncoupled, and the description provided by the relations yields a valid description of the abstraction-group, even without any knowledge about the actual Element instances. On the other hand, this obliges a problem solution to represent both the Elements and their connection in an *abstraction-group*. The C-relations and H-relations create an explicitly formulated skeleton in which, by the means of generation or recognition algorithms, Element instances are created.

A H-relations contains the following information:

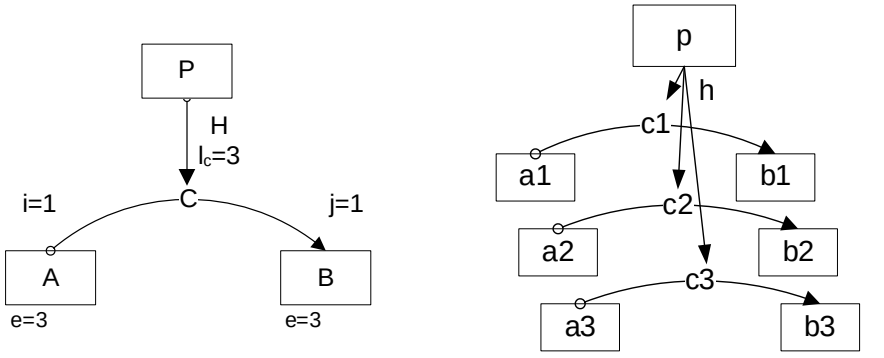
- **Cardinalities**: Each reference from an H-relation class to a C-relation class is associated with a total cardinality l_c. This cardinality models the amount of instances of C-relations in an abstraction-group. In the example shown in Figure 6.9, a C-relation C connects one Element A with one Element B. In order to connect 3 element instances A with 3 element instances of B, three C-relations C are required, which results in $l = 3$.

- **Sequence**: The second type of information stored within the H-relation is called sequence. A sequence is a list of successive relations that in essence resembles a plan for the instantiation order.

- **Constraining rules**: Constrained combination of C-relations by specifying required combinations or not allowed combinations. Constraining rules are formulated in the form of sequence fragments. This type of rules are formulated using references to both C-relations and Elements classes and instances and logic operators.

- **Reference**: Owner-reference to C-relations, owned-reference to an Element.

It is interesting to note that the sequence and constraining rules indicate the managing role the H-relation has over an abstraction-group. As H-relations are



(a) Class representation with $l_C = 3$.

(b) Instance representation with C-relation instances.

Figure 6.9: *H-relations: class and instance.*

always related to parental Element on a one to one basis, they are coupled to cardinalities. According to this, an Element instance can only have one type of H-relation in the topology domain. However, multiple domains of an element can be represented by different types of H-relations.

ACO-Relations

ACO relations represent the non-topology relations of a design problem formulation: **A**nalysis relations, physical **C**oherence constraints and **O**bjective functions. ACO-relations have the following type of information:

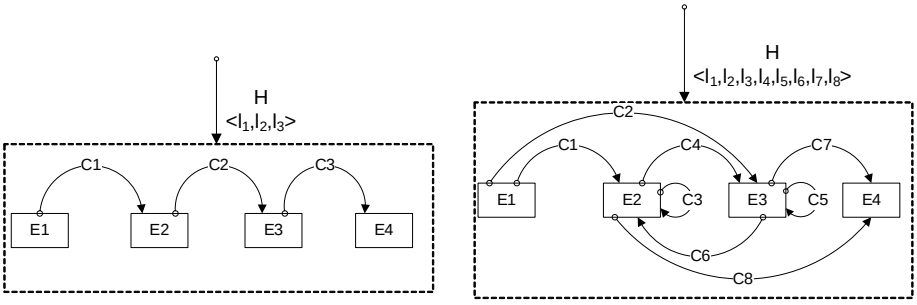
- **References**: owner references are used to parameters declared in Elements, C-relations and H-rations. In this context, cardinalities are also regarded as parameters.
- **Parameters**: model the parameters that not attributed to other elements. For example, an the weight parameters of an objective function.
- **Model**: The model uses references, parameters and operators to describe the nature of the relation.

6.2.3 Types of abstraction-groups

Creating solutions to problem instances consists in instantiating C-relations and Elements in abstraction-groups, which results in a collection of m elements connected by successive $m-1$ C-relations. This collection is regarded as *sequence*, and determines the time dependency of a problem. Abstraction-groups with one

possible sequence, as for example in Figure 6.10(a), are regarded as simple. On the other hand, abstraction-groups allowing multiple distinctive sequences are regarded as complex, as for example the abstraction-group shown in Figure 6.10(b). As consequence, the degree of freedom of a simple abstraction-group only depends on the amount of (unknown) cardinalities, while in a complex one it depends on both the cardinalities and number of possible sequences. The time dependency of this type of problems draws from the fact that the number of elements and C-relations changes as a solution is created, and its final form in unkown until a solution is fully defined. Therefore, an abstractiogroup is complex if: *the number of C-relations n is equal or larger than the number of elements m within the abstraction-group, thus $n \geq m$.*

The example illustrated in Figure 6.10(b) shows a complex abstractiogroup with 4 elements and 8 C-relations. A back coupling occurs at the relations $C3$ and $C5$, which create a loop on a single element. This results in a line-up of the multiple instances of the same element. It has to be pointed out that the addition of one not back coupling relations does not necessarily enlarges the set of sequences. However, the introduction of even one single back coupling relation adds theoretically an infinite amount of sequences to the solution set, since it can be repeated an infinite amount of times. As a result, the example has an infinite set of solutions. Accordingly, the simple group in Figure 6.10(a) has one possible sequences.



(a) Simple abstraction-group.

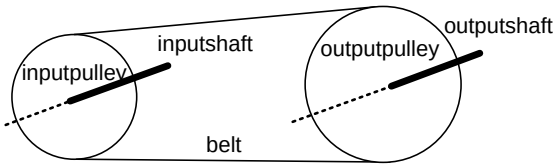
(b) Complex abstraction-group.

Figure 6.10: *Simple and complex abstraction-groups.*

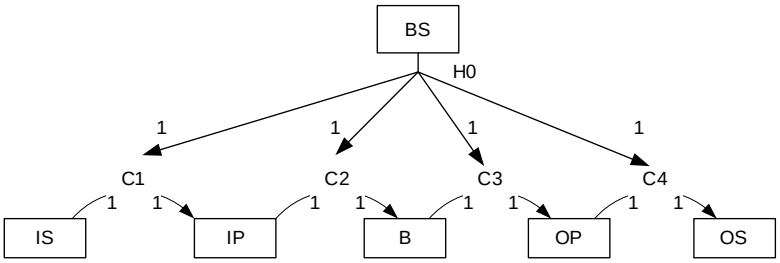
6.3 Example: Belt System Design

The following example illustrates the concepts discussed above. Figure 6.11 shows a sketch of a conceptual belt system. The embodiment of the belt system includes an input shaft (IS), an input pulley (IP), a belt (B), an output pulley (OP) and an output shaft (OS). The components within the embodiment are arranged in order to serve the main functionality of the system, which is transferring force and momentum. This type of problem can be regarded as a topologic design problem, as it concerns the configuration of different types of components. In other words, the topology has to make sense from a functional point of view. This implies that the C-relations are established within the functional domain of transferring energy.

Figure 6.11(b) presents a straightforward topological diagram on the basis of the framework. It is composed of just one level containing all components from Figure 6.11(b) translated into Elements. These Elements are related to each other by the means of four C-relations ($C1$,$C2$,$C3$,$C4$) in such a manner that it resembles the path of the energy flow from input to output. Due to the fact that each component only appears once, all C-relations are one by one relations, with i and j equal to one. A further consequence is that the cardinalities l contained within the H-relation, which is connected to the zero level Element BS (Belt system), are equal to one as well.



(a) A conceptual single belt transmission.

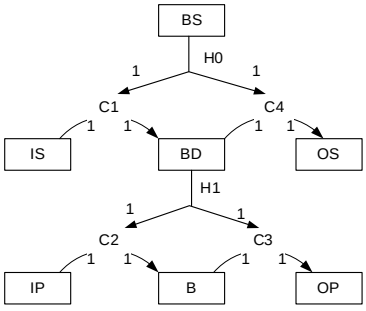


(b) Topological network of the belt system with a single level of detail.

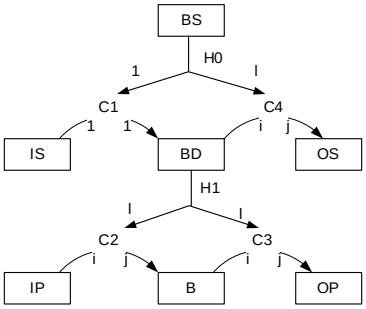
Figure 6.11: *Example of representation of a pulley transmission system.*

The values of the cardinalities are fixed due to the fact that the artifacts'

topology is known. However, the structure of the diagram can be alternated by reorganizing the base Elements into different abstraction-groups. The topology diagram shown in Figure 6.12(a) is constituted by the same physical elements as the structure shown in Figure 6.11(b). However, their representation is done differently, given that the structure in Figure 6.12(b) contains a new Element BD (belt drive), which is composed by the Elements IP,B and OP in a lower level abstraction-group. Although this model looks complexer than the previous one, it allows separating the problem in two independent chunks. By doing so, level 1 is concerned with the search of adequate transmission ratios, while level 2 is concerned with the search of parameter values that satisfy the ratios specified in level 1.



(a) All cardinalities fixed: structure fully defined.



(b) Most cardinalities unknown: structure undefined.

Figure 6.12: *Topological network of the belt system with two levels of detail.*

The TARD model shown in Figure 6.12(a) is topologically fully defined, as its cardinalities are known, and it is limited to one possible sequence of C-relations. On the other hand, the TARD model shown in Figure 6.12(b) is topologically under defined, given that some of its cardinalities are unknown. Depending on the number of unknown cardinalities and its distribution in the TARD network, there are different possible configuration of the solutions. For example, the belt system in Figure 6.11(a) is one alternative. Another possible solution for the under defined situation is the design shown in Figure 6.13, having two belts and output pulleys. This is simply achieved by setting the cardinalities i and j of $C2$ to 1 and 2 respectively, introducing a parallel structure, while the cardinalities at $C3$ are kept to one. It should be pointed out, that due to the parallelism all successive objects in the abstraction-group have parallel instances as well. This is kept consistent by setting the cardinality l for $C3$ in $H1$ equal to 2.

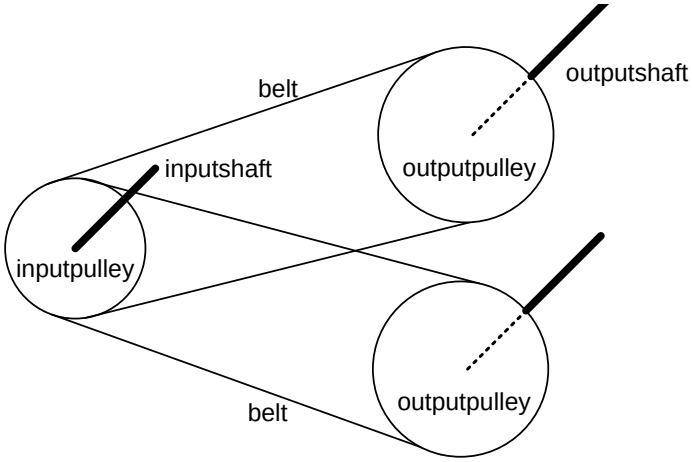


Figure 6.13: *A conceptual double belt transmission: one input, two outputs.*

6.3.1 Proximity Relation

One special case of ACO-relations is the *proximity relation*. Although abstraction-group are in principle treated independently, Elements between different *abstraction−groups* might be physically connected too. Usually, this relationship exist between the first and last Elements of an abstraction-group. These elements are regarded in TARD as boundary Elements. Boundary Elements might be physically connected to components in Elements in other neighboring $abstraction - groups$.

Consider the example in Figure 6.11. Here, the transmission system consists of physically connected parts. Figure 6.12 shows that the Elements of the belt system can be divided into two *abstraction-groups*. Although the output pulley OP and the outputshaft OS are physically connected, they are allocated into different abstraction-group. If there are two instances of Element OP, there have to be two instances of OS as well and vice versa. The resulting physical model is shown in Figure 6.13. The same is true for the Elements IS and IP, however the cardinalities remain 1 in this case. The correlation between those Elements is called proximity relation. In the example, the local cardinalities of IP and OP are set equal to the local cardinalities of IS and OS respectively, resulting in the proximity relations: $e < IP >= e < IS >$ and $e < OP >= e < OS >$, with $\frac{e_1}{e_2} = p$, where p is called proximity constraint. The choice of which Elements to bind by a proximity relation is highly dependent on the problem and its boundary conditions.

Chapter 7

Managing Complexity III: Manipulating Elements

Two techniques to manage problem instance complexity are proposed in this Chapter. One technique manages complexity arising from requirements distribution, while a second technique manages complexity arising from knowledge distribution.

7.1 Introduction

Solving design problem instances can be regarded as a two folded process. Firstly, depending on the distribution of known and unknown entities, a design strategy is developed. Secondly, specialized algorithms instantiate the elements, parameters and relations such that the goal of the design is achieved. While the former process aims at determining the solving order of a problem instance, the latter is concerned with the attribution of values that meet the design goals of the problem. Under the model of complexity presented in Chapter 4, complexity in problem instances appears from uncertainties in defining solving strategies rather than the process of attributing values to design parameters. Therefore, the result of managing complexity of problem instances are methods determining the order (which elements are instantiated when) and directions (e.g. top-down, bottom-up, combinations) in which a design solution can be generated.

This chapter proposes two approaches for managing the complexity of problem instances from the viewpoint of element manipulations. Both approaches are based on the TARD framework. The first approach is a model that relates the cardinalities of the elements with algebraic equations. By doing so, constraint solving algorithms can be implemented to determine the order in which abstraction-groups have to be solved. The second determines the generation ap-

proach as a function of the distribution of design requirements. This method determines which strategies to use for solving an abstraction-group. In other words, the first is concerned with the relation among different abstraction-groups, while the second is concerned with the internal process within an abstraction-group.

Section 7.4 demonstrates the use of both methods in the design of the optical path of an XRF device. Appendix A presents how these methods have been used for an equation generation tool.

7.2 Theory 2: Topology System of Equations

The Topology System of Equations (ToSE) is a set of algebraic equations that link the cardinalities attributed to the building blocks in TARD: e,e_l, i, j, l. These equations can be seen as the bounds keeping Elements, C-relations and H-relations of a design problem tied together. ToSE enables managing both combinatorial and imaginary complexity of problem instances. Combinatorial complexity is managed by incorporating relations among cardinalities that do not change as the design process progresses (not function of time). Imaginary complexity management is achieved by constituting a set of equation that can be used to determine the type of problem: under constrained, over constrained or system of equation. By applying, for example, a constrain solving algorithm, the topologic problem can be solved. ToSE consists of two types of equations, namely, balance equations and vertical equations. While the former describe the relations in one abstraction-group, the latter express the relation among all the abstraction-groups. ToSE also allows determining how changes in one abstraction-group affect other abstraction-groups.

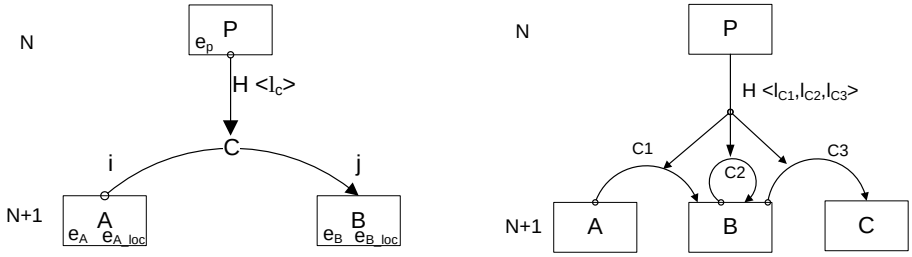
7.2.1 Balance Equations

The example in Figure 7.1(a) shows a simple TARD model composed of one abstract element and one abstraction-group. At the top, the parent element class P at level N is related to the C-relation C in the subgroup at level $N+1$ by the means of the H-relation H. The element classes A and B are connected by C. Calculating the local cardinalities of A and B is in this case simply done by the following equations:

$$e_{A\_loc} = i_c \cdot l_c \tag{7.1}$$

$$e_{B\_loc} = j_c \cdot l_c \tag{7.2}$$

Where i and j are the cardinalities in the C-relation and l denotes the cardinality of C in H. This equation states that the amount of instances of element A is the



(a) Simple abstraction-group.

(b) Complex abstraction-group.

Figure 7.1: *Two abstraction-groups.*

product of the amount of instances of relation C by the amount of instances of element A that have to be connected to a single relation C.

Figure 7.1(b) shows a TARD model that introduces a third element and two C-relations to the previous one. Here, equations 7.1 and 7.2 are no longer valid. Given that there are three relations interfering with this element, another approach is required for calculating the local catrdinalities e_{local}. This can be done by classifying C-relations in two types: incoming CI and outgoing CO. Using this definition, the cardinality of an element e_l can be calculated using either its incoming C-relations or using its outgoing C-relations as follows:

$$e_{local} = \sum j_{CI} \cdot l_{CI} \tag{7.3}$$

$$e_{local} = \sum i_{CO} \cdot l_{CO} \tag{7.4}$$

Given that the cardinalities of the incoming and outgoing relation have to be equal in order to have consistency, both relations can be integrated to result in the equation:

$$\sum j_{CI_E} \cdot l_{CI_E} = \sum i_{CO_E} \cdot l_{CO_E} \tag{7.5}$$

where CI_E and CO_E being the C-relations connected to element E. This equation is regarded as the ***balance equation*** of an element. Balance equations can be assembled for each of the elements in a TARD model.

Consider now the example of the abstraction-group shown in Figure 7.2. The balance equations for element $E2$ is expressed by equation 7.6 and for element $E3$ by equation 7.7. However, elements $E1$ and $E4$ are not related to any incoming or outgoing relations. This is because of the fact that boundary elements are connected to other element in other abstraction-group. Therefore, proximity

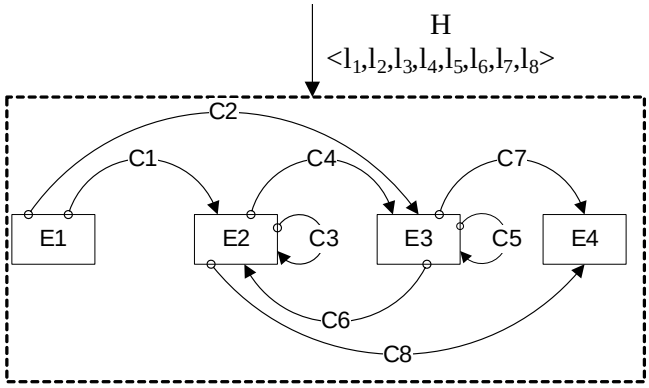


Figure 7.2: *Example of complex abstraction group.*

equations are used to formulate the balance equation of boundary elements. In the case of the first element of an abstraction-group, the contribution of the proximity equation has to be added to the side of the incoming relations. In the case it is the last element, the proximity equation is added to the side of the outgoing relation. Following these remarks, the balance equations for the elements $E1$ and $E4$ are equation 7.8 and equation 7.9, respectively.

$$j_{C1} \cdot l_{C1} + j_{C3} \cdot l_{C3} + j_{C6} \cdot l_{C6} = i_{C3} \cdot l_{C3} + i_{C4} \cdot l_{C4} + i_{C8} \cdot l_{C8} \tag{7.6}$$

$$j_{C2} \cdot l_{C2} + j_{C4} \cdot l_{C4} + j_{C5} \cdot l_{C5} = i_{C5} \cdot l_{C5} + i_{C6} \cdot l_{C6} + i_{C7} \cdot l_{C7} \tag{7.7}$$

$$e_{E1}^{prox} = i_{C1} \cdot l_{C1} + i_{C2} \cdot l_{C2} \tag{7.8}$$

$$j_{C7} \cdot l_{C7} + j_{C8} \cdot l_{C8} = e_{E4}^{prox} \tag{7.9}$$

Where e_E^{prox} denotes the proximity relation of element E

7.2.2 Vertical Equations

Vertical equations are used to relate cardinalities across abstractiongroups in a TARD model. To explain the concept of vertical equations, consider the TARD model shown in Figure 7.1(b). Here, for each instance of the element P exists one corresponding instance of abstraction-group containing instances of A and B.

In this case, the global cardinalities of the elements A and B keep track of the amount of element instances in all abstraction-groups, which can be expressed by the following expression:

$$e_B = e_P \cdot \sum j_{CI} \cdot l_{CI} \tag{7.10}$$

$$e_B = e_P \cdot \sum j_{CO} \cdot l_{CO} \tag{7.11}$$

The equation for calculating the global cardinality of an element is based on level N and level $N+1$. The equation for element P (e_p) can be derived in the same fashion, relating the to the level $N-1$. By following this pattern until the zero-element (level $N-N$), the equations connect all related elements successively across from level N+1 until the zero level into a single relationship. The resulting equation, is regarded as the elements vertical equation. The vertical equation states that a change in the value of an element's cardinality on a high level has a direct effect on the cardinality of an element on a lower level and vice versa. A general expression of a vertical equation can be written as follows:

$$e_{E_N} = \prod_{m=N}^{m=1} \sum \left[(i,j)_{C_{I;O}} \cdot l_{C_{I;O}} \right]_m \tag{7.12}$$

Where e_{E_N} denotes the global cardinality of element Eon level N. Either incoming or outgoing C-relations can be used.
Figure 7.3 shows an example of a TARD model with two levels. The lines indicate the how the cardinalities of different elements are related by vertical equations. In this example, two alternative but equally valid vertical equations for element F are:

$$e_F = \left(j_{C_{E-F} \cdot l_{C_{E_F}}} \right) \cdot \left(j_{C_{A-B} \cdot l_{C_{A_B}}} \right) \tag{7.13}$$

$$e_F = \left(i_{C_{F-G} \cdot l_{C_{F_G}}} \right) \cdot \left(i_{C_{B-C} \cdot l_{C_{B_C}}} \right) \tag{7.14}$$

The path used for formulating Equation 7.13 is indicated by the dotted line in Figure 7.3, while the path used for equation 7.14 is indicated by dashed line. Two more valid equations can be formulated:

$$e_F = \left(j_{C_{E-F} \cdot l_{C_{E_F}}} \right) \cdot \left(j_{C_{B-C} \cdot l_{C_{B_C}}} \right) \tag{7.15}$$

$$e_F = \left(i_{C_{H-I} \cdot l_{C_{H_I}}} \right) \cdot \left(i_{C_{A-B} \cdot l_{C_{A_B}}} \right) \tag{7.16}$$

In contrast, element H only has one possible vertical equation:

$$e_H = \left(i_{C_{H-I} \cdot l_{C_{H_I}}} \right) \cdot \left(j_{C_{C-D} \cdot l_{C_{C_D}}} \right) \tag{7.17}$$

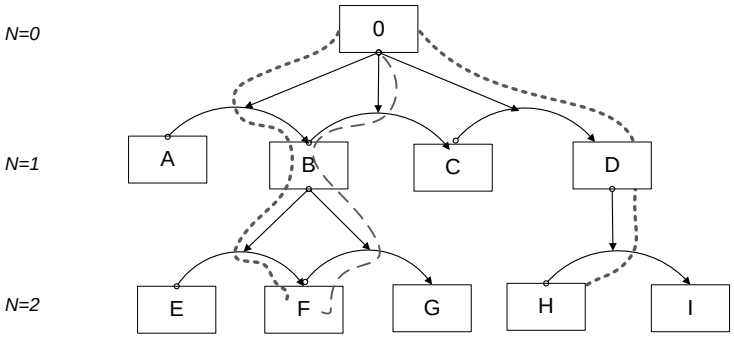


Figure 7.3: *Relational paths of vertical equations in a two level TARD model.*

As it can be seen, the number of valid vertical equations of a given element depends on the incoming and outgoing C-relations in the path. The purpose of the vertical equations is to relate the model mathematically across the levels of detail. In order to take all elements into account by vertical equations, it is sufficient to formulate one vertical equation for each base element in a TARD diagram. For the example in Figure 7.3 this means that abstract elements B and D are already covered by the equations of the elements E, F, G, H and I. In this example, 7 vertical equations are sufficient to relate all abstraction-groups.

7.3 Method 4: The Local Grammar Method

As described in Chapter 4, a problem instance is the result of setting embodiment, scenario and performance requirements. Requirements are set by instantiating elements, relations, or parameters of the problem class. Depending on which entities are instantiated in one abstraction-group, two different types of complexity arise:

- ***Time-dependent periodic complexity***: is the uncertainty of knowing how many and in which order the C-relations are instantiated to produce a problem solution. This type of complexity in attributed in TARD to complex abstraction-groups, described in Subsection 6.2.3.

- ***Time-independent imaginary complexity***: is the uncertainty of knowing if solutions are created bottom-up, top-down manner or as combinations of both. In TARD, this type of complexity is found in problems where re-

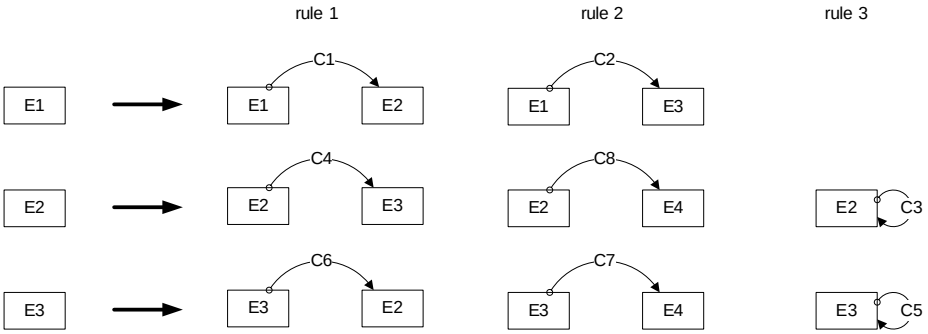


Figure 7.4: *Local grammar rules for the elements in Figure 6.10(b).*

quirements are set by instantiating elements at different levels of abstraction. This type of complexity is independent of the type of abstraction-group (complex or simple).

The local grammar method aims at managing these complexities by presenting a formalism for grammar generation that depends on the allocation of the design requirements.

7.3.1 Grammar Rules and their Application

In order to apply grammar theory in TARD models, a C-relation class relating two element classes is considered as a grammar rule. The advantage of C-relations is that it can represent various grammar rules at once: IC instantiated and OC not instantiated, IC and OC not instantiated, and OC-instantiated and IC instantiated. As the instantiation of C-relations depend on the direct neighbors of an element, this procedure is referred to as the *local grammar method*.

Figure 7.4 shows the local rules of the elements $E1$, $E2$ and $E3$ in Figure 7.2. Generating a network of components starts by applying one of the two rules available for element $E1$. If rule 2 is chosen, an instance of $C2$ that connects element $E1$ to $E3$ is created. Done this, the algorithm continues by checking which rules are available to connect element $E3$ and then selecting one rule. This process continues iteratively, until an end element (here $E4$) or end criterion is reached, with a sequence as a result. Specialized algorithms can be used to drive the search process.

7.3.2 Adding Complementary Rules

Complimentary rules can be defined within an H-relation to constrain the generation of sequences. These rules are used to model design rules and reduce the

solution space of a problem. For example, for the case of CSIM design, complementary rules can be used to model how expert designers alternate between different types of cooling channels when constructing cooling circuits.

When a sequence fragment matches an already made sub-sequence, the complementary rule is applied. This is particularly useful to constrain the generation of known infeasible combinations and the infinite usage of rules. For example, a repetition of a certain element can be constrained in this way, by defining a rule, stating that after a C-relation has been used for a given number of times, it has to be excluded from the following set of local grammar rules. Figure 7.5 shows such a rule, where $C2$ is excluded from the local grammar iteration after using it twice. As a result, three instances of E are chained up and $C3$ remains as the only choice for the next iteration.

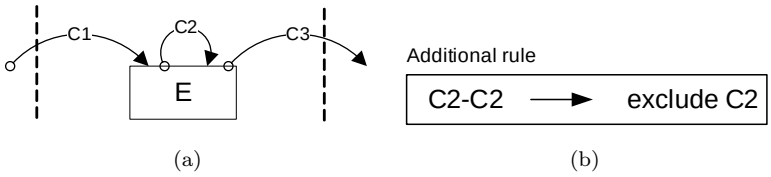


Figure 7.5: *Example of an adding a grammar rule in the local grammar method.*

7.3.3 Guiding the Search Process

A sequence has been described as one possible combination of the elements in an abstraction-group. This combination is stored in the form of a list of successive C-relations (a string), which starts and finishes with an C-relations. In order to manage the generation of sequences, a factor k_C associated to each C-relation C instance in the string has been defined. This factor denotes the amount of relation instances at a given position in the sequence. Thus, the total amount of instances l_C of the C-relation in a sequence C is given by the sum of its factors k_C. By relating both terms, an expression termed as sequence equation is derived to aid the management of complex abstraction-groups:

$$l_c = \sum K_c \tag{7.18}$$

In a complex abstraction-group, algorithms iteratively connect elements using C-relations. While this is done, a sequence is generated and expanded, which in turn is used as guidance by the complementary rules. By adding each of the factors k_C, a comparison can be made to the sequence equations. This enables making predictions over which relation to choose in the next iteration step or, if no alternative can be found, to stop. Thus, the sequence equation provides a guidance to the assembly processes in order to end up with a consistent sequence.

7.3.4 Creation vs. recognition

Within one abstraction-group, the concepts of top-down and bottom-up are related to mechanisms used in the instantiation of Elements and C-relations . In top-down approaches, instantiated upper layers determine the creation of the lower level. For example, an instantiated C-relation determines the instantiation of an element. In a bottom-up approach occurs the opposite: instantiated lower layers are used to recognize upper non instantiated levels. For example, recognize if two instantiated elements can be related by one of the C-relations in the abstraction-group. Determining which of these processes -creation or recognition- is to be applied is a function of which elements and relation have been previously instantiated. Its instantiation is the consequence of how the requirements are distributed in the problem class.

The left side of Figure 7.6 shows an example of an abstraction-group described by the element classes A, B and C, while the right side shows a pool of element instances a, b and c. The number of instances of each type in the pool is represented by X_a, X_b and X_c respectively. Furthermore, the amount of instances required in a solution is denoted by the element cardinalities. The number of element instances that result from a recognition process or a creation process is denoted as f_x and f_g, respectively. Element cardinalities that exclusively result from a generation process are given by $e = f_g$ and those resulting from recognition are given by $e = f_x$. In order to determine which approach should be used, the following relations have been determined:

- System type 1: A creation mechanism is used, as no elements have been instantiated yet. It has the following characteristics:

$$x = 0, e = f(g)$$

- System type 2: Instances are only created for element classes that have no existing instances. Recognition is used to determine how to connected existing instances. It has the following characteristics:

$$x > 0, x < e, e = f(g)$$

- System type 3: Instances are only created for element classes that have no existing instances. C-relations are recognized for instantiated elements, and created for non instantiated elements. It has the following characteristics:

$$x > 0, x < e, e = f(g) + f(x)$$

- System type 4: C-relations are only recognized. It has the following characteristics:

$$x > 0, X = e, e = f(x)$$

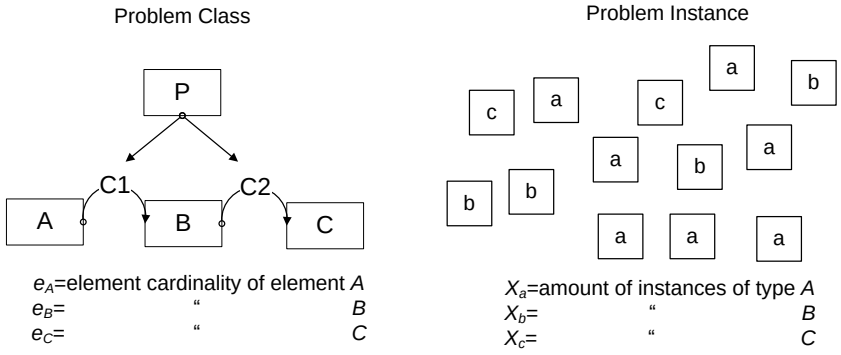


Figure 7.6: *Class and instance representations of an abstraction-group.*

7.4 Example: XRF Optical Path Design

This example deals with the design of an optical chamber of an x-ray fluorescence spectrometer (XRF spectrometer), described in [57]. An XRF spectrometer is an instrument capable of determining the chemical composition of material samples. This is done by radiating a sample with a high-energy x-ray. The radiation causes the sample to expel photons which are absorbed by an energy dispersive detector. By analyzing the wave lengths of the energy absorbed by the detector, different material compositions can be identified. The arrangement of the components in the optical chamber of an XRF spectrometer are shown in Figure 7.7, consisting of: the x-ray tube, the sample, the detector and some diaphragms.

7.4.1 TARD Model

The function of the device is primarily determined by the path of the radiation from the x-ray tube to the detector. Therefore, the components in the TARD are arranged according to the path followed by the beam, as this one determines the flow of energy in the artifact. For this example, components like the casing or electrical circuits are not taken into account, since they are allocated in different functional domains. The five base element classes are: X-ray tube, diaphragm type 1, sample, diaphragm type 2 and detector. As consequence, only one abstraction-group is defined. The C-relations are formalized according to the fashion in which the element functions act upon each others (as described in Subsection 5.3.1). The H-relation relates the group of C-relations to the zero level element "optical chamber". All proximity relations equal one. The resulting

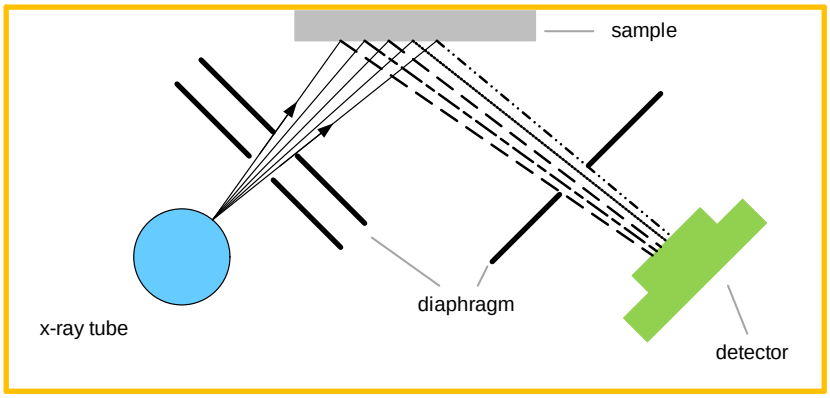


Figure 7.7: *Schematic of the optical path design of an XRF spectrometer.*

TARD model is shown in Figure 7.8. This model represents the problem class of the problem.

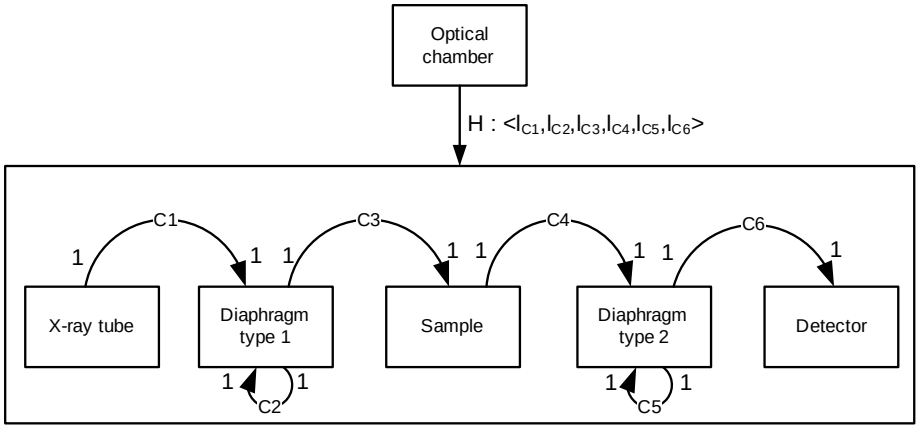


Figure 7.8: *TARD model of XRF optical path design.*

7.4.2 Assembling ToSE

To construct the problem instance, requirements are set by specifying which elements are instantiated and which cardinalities are known. The global element cardinalities of the x-ray tube, the sample and the detector are set equal to one (1), given that only one of each component is used in the optical chamber. Ad-

ditionally, as each component is only related to one other component, the values of the cardinalities i and j of all C-relation equals one. At the hand of the information gathered in the diagram, the vertical and balance equations are now formulated. Given that all elements are base elements, the ToSE of this problem consists of 5 vertical and 5 balance equations.
The vertical equations are:

$$e_{tube} = [i_{C1} \cdot l_{C1}] \cdot e_0 \Longrightarrow 1 = [1 \cdot l_{C1}] \cdot 1 \Longrightarrow l_{C1} = 1$$

$$e_{dia1} = [j_{C1} \cdot l_{C1} + j_{C2} \cdot l_{C2}] \cdot e_0 \Longrightarrow e_{dia1} = [1 \cdot 1 + 1 \cdot l_{c2}] \cdot 1 \Longrightarrow e_{dia1} = 1 + l_{C2}$$

$$e_{sample} = [j_{C3} \cdot l_{C3}] \cdot e_0 \Longrightarrow 1 = [1 \cdot l_{C3}] \cdot 1 \Longrightarrow l_{C3} = 1$$

$$e_{dia2} = [j_{C4} \cdot l_{C4} + j_{C5} \cdot l_{C5}] \cdot e_0 \Longrightarrow e_{dia2} = [1 \cdot 1 + 1 \cdot l_{c5}] \cdot 1 \Longrightarrow e_{dia2} = 1 + l_{C5}$$

$$e_{detector} = [j_{C6} \cdot l_{C6}] \cdot e_0 \Longrightarrow 1 = [1 \cdot l_{C6}] \cdot 1 \Longrightarrow l_{C6} = 1$$

The balance equations are:

$$e_{tube}^{prox} = i_{C1} \cdot l_{C1} \Longrightarrow 1 = 1 \cdot l_{C1} \Longrightarrow l_{C1} = 1$$

$$j_{C1} \cdot l_{C1} + j_{C2} \cdot l_{C2} = i_{C2} \cdot l_{C2} + i_{C3} \cdot l_{C3} \Longrightarrow l_{C2} = l_{C2}$$

$$j_{C3} \cdot l_{C3} = i_{C4} \cdot l_{C4} \Longrightarrow 1 = 1$$

$$j_{C4} \cdot l_{C4} + j_{C5} \cdot l_{C5} = i_{C5} \cdot l_{C5} + i_{C6} \cdot l_{C6} \Longrightarrow l_{C5} = l_{C5}$$

$$e_{detector}^{prox} = j_{C6} \cdot l_{C6} \Longrightarrow 1 = 1 \cdot l_{C6} \Longrightarrow l_{C6} = 1$$

In this example, the balance equations cannot be used to solve implicit cardinalities. However, they confirm the consistency of the model. There are two unknown cardinalities values subject to design.

7.4.3 Generating Sequences

In this example, the recursive relations $C2$ and $C5$ can be repeated an arbitrary amount of times in the process of generating a sequence. By defining an additional grammar rule in the H-relation, the number of C-relations can be constrained. For example, if the maximum number of diaphragms type 1 is considered to be three, a maximum of two $C2$ can be used. Figure 7.9 shows two complementary rules, one regarding the C-relation $C2$ and another regarding the C-relation $C5$.

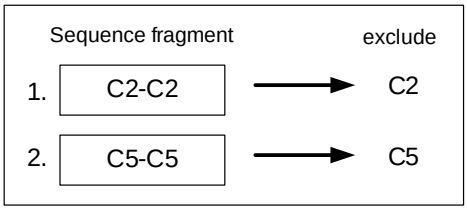


Figure 7.9: *Complementary grammar rule in XRF design.*

In this example, no instances are provided as requirements in the problem instance. Therefore, it is considered to be a type 1 problem (creation, as described in Subsection 7.3.4). The problem is solved by applying the local grammar method. Figure 7.10 shows an example of the steps followed for generating a possible sequence. The figure shows which are the possible rules to use, and which rules where selected, for each generation step. Which rule to select should be performed by a decision making algorithm. The latter being either a simple random selection algorithm or a sophisticated algorithm, like for example genetic algorithm. In either case, the sequence is expanded at each iteration step until the process reaches the last element or an end criteria based on the design performances.

The resulting sequence for the abstraction-group determines the remaining two degrees of freedom by adding two sequence equations to the system of equations, one for $C2$ and one for $C5$:

$$l_{C2} = \sum k_{C2} = 1 + 1 = 2$$

$$l_{C5} = \sum k_{C5} = 0$$

By substituting these equations in the vertical equations, the cardinalities can be calculated:

$$e_{dia1} = 1 + l_{C2} = 1 + 2 = 3$$

$$e_{dia2} = 1 + l_{C5} = 1 + 0 = 1$$

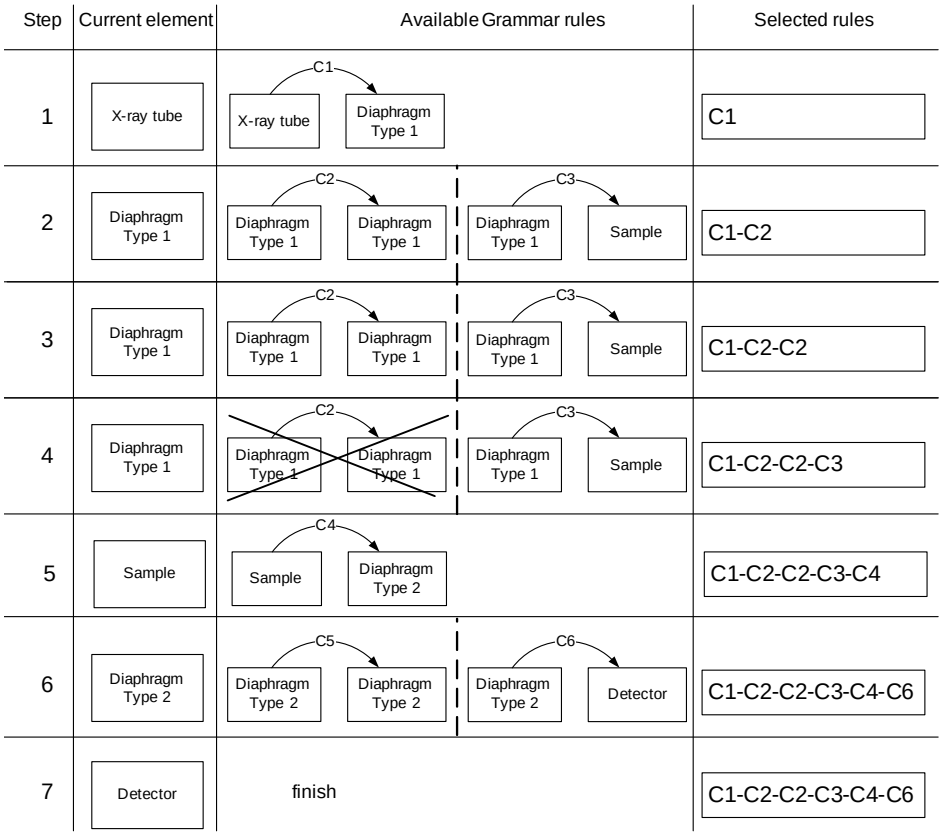


Figure 7.10: *Example generation of a sequence for the XRF optical Path design.*

Chapter 8

Managing Complexity IV: Manipulating Parameters

This chapter presents a method for determining appropriate solving orders of networks of parameters in problem instances. In terms of ADT, the algorithm determines the order in which the DM of a problem instance can be solved.

8.1 Introduction

As it was described in Chapter 7, time-independent imaginary complexity in problem instances originates from uncertainties in knowing the order in which DPs can be solved. When the problem has many DPs, choosing a solving path at the hand of the problem's DM is not straight forward. While Chapter 7 described methods for determining how to instantiate elements, this chapter focuses on parameters related by analysis, physical coherence constrains, and knowledge rules relations. These relations can be explicitly written as mathematical models, but can also be graphically represented using knowledge graphs, as proposed by [57]. In this chapter, these directed graphs are used to represent the network of parameters and relations, and explain the basis of the algorithm.

8.1.1 Knowledge Graphs (KG)

Knowledge Graphs represent the network of deterministic knowledge and the possible paths through the network to resolve all parameters and find a solution. Knowledge graphs (KG) consist of nodes and directed edges. Nodes represent parameters and the directed edges describe the relations among each parameter. The directions of the edges are pointed toward the parameter to resolve. For example, if node D has N ingoing edges, the parameter represented by node D

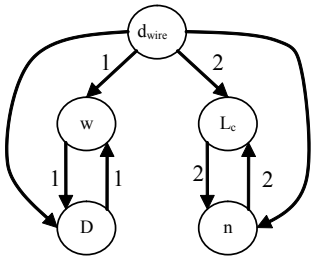


Figure 8.1: *Knowledge graph of equation 8.1 and equation 8.2.*

requires N other parameters to be known in order to resolve it. The knowledge graphs edges are labeled according to the relations to which they belong. An example of such a knowledge graph is shown in Figure 8.1. This graph represents equation 8.1 and equation 8.2, used in the design of a compression spring. The first equation describes the winding ratio w as a fraction of the spring diameter D and the wire diameter d_{wire}. The second describes the maximum spring compression L_c as function of the number of coils n and the wire diameter d_{wire}.

$$w = \frac{D}{d_{wire}} \tag{8.1}$$

$$L_c = n \cdot d_{wire} \tag{8.2}$$

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 8.1, 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 equation (1) is not guaranteed to be exactly a DIN value. Therefore, d_{wire} has to be resolved by another rule before equation (1) can be used to resolve either w or L_c.

8.2 Method 5: KGM Solving Algorithm

The KGM algorithm aims at determining orders in which parameters related by ACO-relations can be solved. It does so at the hand of two complexity measures that can be calculated for each of the parameters in the problem, namely, the effort E and the influence *Inf*. The effort measures the number of parameters that have to be known in order to resolve a parameter with a certain equation. The influence measures the number of parameter that can be solved by solving this parameter. Furthermore, the algorithm is also based on four states a parameter can have: unknown, known, driver and driven. At the beginning of the problem,

all parameters have the state unknown. Then, as the designers set requirements on the problem, some parameter take the state known. A driver state corresponds to parameters that are instantiated by an algorithm (e.g. randomly) or by a design decision, while a driven state defines a parameter that can be calculated in a relation where the values of all other parameters are known. By assessing the values of the effort and the influence of each parameter, the algorithm chooses which parameter should become driver and which becomes driven as consequence of that decision.

8.2.1 Knowledge Graph Matrix (KGM)

The first step in the method consists in assembling the Knowledge Graph Matrix (KGM) of the problem. The adjacency matrix of a directed **G** on n vertices is the $n \times n$ matrix where the non diagonal entry a_{ij} is the number of edges from vertex i to vertex j, and the diagonal entry a_{ii} is number of loops at vertex. However, given the form of knowledge graphs (KG), no loops are present and therefore the diagonal entries remain empty. Furthermore, each entry in the matrix is accompanied by the equation id related to that entry. For the case of the knowledge graph in Figure 8.1, the corresponding adjacency matrix would take the following form:

$$KGM = \begin{pmatrix} & d_{wire} & D & w & L_c & n \\ d_{wire} & 0 & 1(1) & 1(1) & 1(2) & 1(2) \\ D & 0 & 0 & 1(1) & 0 & 0 \\ w & 0 & 1(1) & 0 & 0 & 0 \\ L_c & 0 & 0 & 0 & 0 & 1(2) \\ n & 0 & 0 & 0 & 1(2) & 0 \end{pmatrix} \tag{8.3}$$

8.2.2 Effort and Influence

Two measures are used to assess how each node of the knowledge graph is related to the others. These two measures are used to determine which parameters are "easier" to solve and which will help solving most of the left unknown parameters. The first case corresponds to the parameter with the lowest effort, while the second corresponds to those with the largest influence.

Effort

Each node, or parameter, has an effort (E) for each of the relations in which it is included. The effort represents the number of parameters that have to be known in order to resolve a parameter with a certain equation. In a knowledge graph, the effort in one relation of one parameter corresponds to the sum of all the edges pointing towards that node.

Using the matrix representation, the effort of a parameter p in a relation n is calculated by:

$$E_{p,n} = \sum_{i=1}^{l} KGM\,(i,p) \tag{8.4}$$

where l equals the number of rows of the KGM. Furthermore, if one column of the matrix has entries that relate this one to others, the effort of the corresponding parameter is set to 1,which represents the effort the parameter requires for solving itself.

Influence

Each node, or parameter, has an influence (I) that measures the number of relations in which this parameter is present. This equals the sum of all relation present in the row corresponding to a given parameter and is calculated as follows:

$$In_p = \sum_{i=1}^{l} KGM\,(p,i) \tag{8.5}$$

where l equals the number of columns of the KGM. If one row does not contain entries, the parameter related to it has an influence equal to zero. This means that this parameter cannot be used to solve others.

Parameter	d_{wire}	***D***	***w***	***n***	***Lo***
Effort	1	2(1)	2(1)	2(2)	2(2)
Influence	4	1	1	1	1

Table 8.1: *Efforts and influences at problem class.*

In Table 8.1 the efforts and influences of the KGM shown in equation 8.3 are presented. As the table shows, the parameters *D, w, n* and *Lc* have all effort equal to 2 and influence equal to 1, while the parameter d_{wire} has an effort equal to 1 (which is the effort of solving itself) and an influence equal to 4. This suggest that for this case d_{wire} is the easiest variable to solve as well as it is the parameter that if solved will help the most parameters to be solved.

8.2.3 Parameter States

In a design problem, a parameter can have two states, namely, known and unknown. In the first case, the parameter has been instantiated, and its value imposes a requirement on the problem. The second case corresponds to the parameters the designer is interested in solving.

A parameter can become known according to two possibilities:

1. As a design decision of the designer or, in the case of automated design, by an algorithm (e.g. randomly).

2. By calculating it from with a relation where precisely one parameter is unknown.

In this method, these cases are regarded as two more states possibilities of a parameter. The first one is termed as driver state, as the parameter being instantiated will drive the calculation of other unknown parameter. The second case is termed driven state, as the instantiation is driven by the previous algorithmic instantiation of other parameters. According to this, the four states a parameter can have in this approach are: known, unknown, driver and driven.

In these terms, the goal of solving a design problem is to transform parameter with unknown states into known, driver and driven states such that valid solutions are found or a certain objective function is maximized or minimized. It is important to differentiate between the optimization or constrain solving problem and the one this method aims at. As this is concerned with defining the order in which parameter have to be instantiated, the optimization or constrain solving algorithm for solving the values of the parameters are not further studied. The three possible types of state transition are:

1. Unknown-known: occurs when the problem statement is being made by attributing a value to one parameter as a requirement the design has to satisfy. Is attributed to the change from problem class to problem instance.

2. Unknown-driver: occurs by attributing a value to a parameter according to a design decision or an algorithmic step (e.g. by randomly attributing a value). Is attributed to the change from problem instance to problem solution.

3. Unknown-driven: occurs when the effort of a parameter becomes zero after KGM recalculation. This means that this variable can be calculated by using the relation attributed to the effort that became zero. Is also attributed to the change from problem instance to problem solution.

8.2.4 KGM Transformations

As it will be explained later, each time a state transition occurs, the KGM changes and has to be recalculated. Recalculating the KGM is performed by:

1. Eliminating all entries in the row and the column that corresponds to that parameter from the KGM.

2. Recalculating all efforts and influences according to equations 8.4 and 8.5.

Parameter	d_{wire}	***D***	***w***	***n***	***Lo***
Effort	1	0	1(1)	2(2)	2(2)
Influence	3	0	0	1	1

Table 8.2: *Efforts and influences for problem instance with D known*

Consider the example of the KGM in equation 8.3. If one assumes that the parameter D (spring diameter) is set as known and all the others are unknown, the KGM is transformed as shown in equation 8.6, obtaining the efforts and influences shown in Table 8.2.

$$KGM = \begin{pmatrix} & d_{wire} & D & w & L_c & n \\ d_{wire} & 0 & 0 & 1(1) & 1(2) & 1(2) \\ D & 0 & 0 & 0 & 0 & 0 \\ w & 0 & 0 & 0 & 0 & 0 \\ L_c & 0 & 0 & 0 & 0 & 1(2) \\ n & 0 & 0 & 0 & 1(2) & 0 \end{pmatrix} \tag{8.6}$$

8.2.5 Identifying Driver and Driven

The essence of the KGM algorithm resides in choosing drivers and identifying which parameter becomes driven as consequence of the that decision. Drivers are identified by choosing the parameter with the lowest effort. If the value of the minimum effort is shared by more than one parameter, then the parameter with the highest influence among them is chosen. By doing so, it is avoided that two parameter become over constrained as the algorithm progresses. This also minimizes the effort of other parameters. In the case that the minimum effort and the highest influence are shared by more than one parameter, either one of the parameters can be chosen as driver parameter.

Consider the efforts and influences in Table 8.2. As it can be seen, the parameter d_{wire} has the lowest effort. As no other parameter has an effort equal to that of d_{wire}, this parameter becomes automatically a driver parameter.

Drivens are identified as those parameters whose effort becomes 0 (zero) after a KGM transformation has taken place. In this case, this parameter can be calculated by using the relation corresponding to the effort that became 0 (zero).

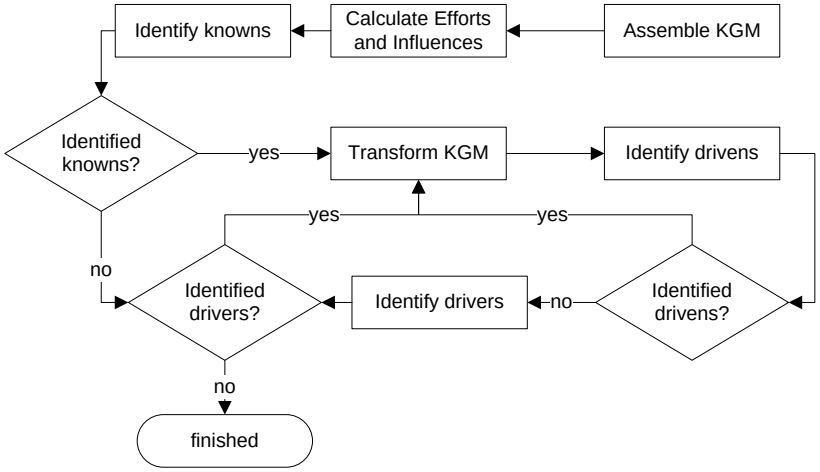


Figure 8.2: *KGM solving algorithm.*

8.2.6 The Algorithm

Having presented the KGM, its transformations and a method for identifying drivers and drivens, the algorithm for recognizing the sequence in which to solve a parametric design problem is presented in Figure 8.2.The steps in the algorithm are the following:

1. Make a knowledge graph including all relations and parameters.
2. Assemble the KGM and calculate efforts and influences.
3. Identify which parameters have been attributed as known and transform the KGM.
4. Identify if any variable has changed it state from unknown to driven. If this is the case, perform a KGM transformation. This step is continued until no parameter change to driven state.
5. Identify one driver parameter and transform the KGM.
6. Repeat steps 3, 4 and 5 until all parameters are changed to the state driven, driven or known.

Table 8.3 shows the results of applying the algorithm on the example shown in Table 8.2. As the table shows, the state transition occurs when setting the value of D to known in step 1. In step 2, the recalculated efforts and influences are shown. Here, the variable d_{wire} is identified as the best to be changed into driver.

Table 8.3: *Result of KGM transformations in example.*

Parameter		d_{wire}	***D***	***w***	***n***	***Lo***
Step 1	Effort	1	2(1)*	2(1)	2(2)	2(2)
	Influence	4	1	1	1	1
Step 2	Effort	1**	0	1(1)	2(2)	2(2)
	Influence	3	0	0	1	1
Step 3	Effort	0	0	0***	1(2)	1(2)
	Influence	0	0	0	1	1
Step 4	Effort	0	0	0	1(2)**	1(2)
	Influence	0	0	0	1	1
Step 4	Effort	0	0	0	0	0***
	Influence	0	0	0	0	0

State transition: * unknown-known, ** unknown-driver, *** unknown-driven

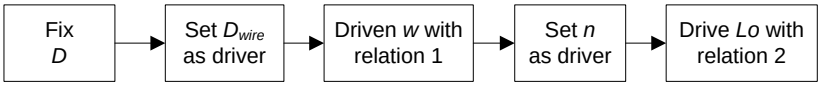


Figure 8.3: *Example of strategy.*

It can also be seen that the variable w gets an influence of zero, as this parameter cannot influence solving others. In step 3, the recalculated KGM measurements indicate that w becomes driven, as its effort drops to zero. As w is only contained in relation 1, its effort only depends on this equation. Therefore, the value of w can be calculated by using equation 1 in combination with the previously defined values of D and d_{wire}. In step 2, the parameter n is chosen as driver, as no other parameter became driven. However, parameter L_o could have also be chosen, as its effort and influence values equals the one of n. As result of a new KGM transformation, parameter L_o becomes driven with equation (2).

The result of applying the algorithm is a sequence of proposed parameters state transitions. The sequence can be outlined by keeping track on the order in which parameter states change from unknown to known, drivers and driven. In the case of driven, the relations through which they are calculated have to be recorded too. After this algorithm is applied to a given parametric design problem, the resulting sequence can be used by a constraint solving or optimization algorithm to search for the numerical solution of the problem. For the example in Table 8.3, the resulting sequence is shown in Figure 8.3.

8.3 Benchmarking

In [30] a survey of decomposition methods for constraint systems is presented. Here, decomposition methods have been investigated from different perspectives, and compared against each others. One consists in identifying subsystem as an interesting (i.e., solvable) subsystem. A second approach aims at identifying points of weakness in the constraint system for partitioning the problem. A third one consists in identifying subsystems as solvable provided that the complementary subsystem is solvable as well. The fourth, denominated recursive division methods, works by iteratively splitting the constraint system into components, themselves subject to further splitting. The review finishes by proposing two properties that are desirable to meet real-life applicative requirements: generality and reliability.

This algorithm differentiates from the ones presented in [30] as follows:

1. The directed graph consists of nodes denoting the parameters, and arcs denoting its relations, while the adjacency graph of the methods in [30] is composed of nodes that represent both the parameters and the relations. Arcs are used to describe which parameters are involved in which relations. The adjacency matrix of this graph allows solving asymmetric problems. Asymmetric problems are those in which at least one relation contains a parameter that is not solvable as function of the others.

2. The decomposition is based on two complexity measurements, namely, the effort and the influence. Because of the generality of the approach, the algorithm can be used in different types of CS problems, as for example, geometric and topological problems.

3. The KGM algorithm does not recognizes systems of equations, and is therefore limited to be used in under-constrained problems. Other algorithms discussed in [30] do have the capability of recognizing systems of equations and over-constrained problems.

8.4 Example: Compression Spring

The parametric design of a compression spring is used to demonstrate the algorithm at the hand of one combination of known and unknown parameters. A compression spring is modeled as a single element with 11 parameters and 6 relations (material is kept constant). The parameters are stated in Table 8.4 and the relations in equation 8.7 - 8.12. An ID has been attributed to each relation. Figure 8.4 shows the KG of this example with the IDs of each relation appearing along the arcs. At the hand of the KG, the KGM is assembled and the efforts and influences of each parameter are calculated, as shown in Table 8.5 and 8.6 respectively.

Table 8.4: *Parameters considered in compression spring design.*

Parameter	***Description***
D	mean diameter
d	wire diameter
n	number of coils
A	space between coils
L0	uncompressed length
Ls	compressed length
Lc	maximum compressed length
s	compression
sc	maximum compression
R	spring constant (stiffness)
F	force at compression

$$R = \frac{G}{8} \cdot \frac{d^4}{D^3 \cdot n} \mapsto ID = 1 \tag{8.7}$$

$$F = R \cdot s \mapsto ID = 2 \tag{8.8}$$

$$L_0 = s + \ L_s \mapsto ID = 3 \tag{8.9}$$

$$L_0 = sc + \ L_c \mapsto ID = 4 \tag{8.10}$$

$$L_c = n \cdot d \mapsto ID = 5 \tag{8.11}$$

$$L_0 = n \cdot (A + d) \mapsto ID = 6 \tag{8.12}$$

The combinations of known and unknown that have been used to demonstrate the algorithm is shown in Table 8.7.

In Table 8.8 the efforts and influences that result from applying the algorithm are presented, while Figure 8.5 shows the resulting sequence. As it can be seen, some parameters have more than one effort value attributed. This is the consequence of having some parameters present in more than one equation. The table also shows that the parameter *sc* becomes driven with equation 8.10 after recalculating the KGM for the known parameters. This is due to the fact that all parameters in relation 8.10, except *sc*, have an attributed value. In step 4, it can be seen that although the parameters *s, R* and *Ls* have the same minimum value of the effort (effort = 1), the parameter *s* is chosen as driver. The reason for doing so is that the influence of *s* is larger than that of the parameters *R* and*Lo*, which means that attributing a value to this parameter will reduce the effort of two other parameters, speeding the course of the algorithm.

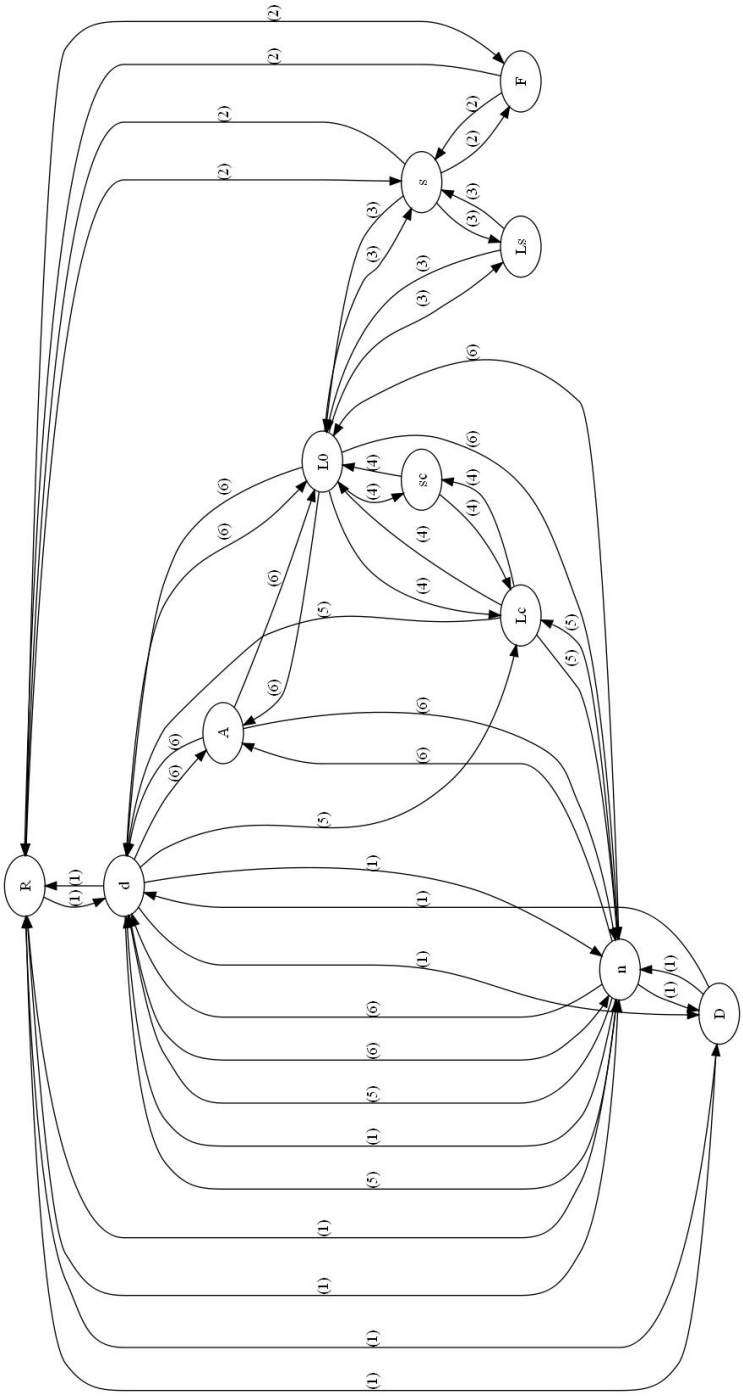


Figure 8.4: *Knowledge graph of spring design.*

Table 8.5: *KGM of compression spring design.*

	D	**d**	**n**	**A**	**Lo**	**Ls**	**Lc**	**s**	**sc**	**R**	**F**
D	1(1)	1(1)							1(1)		
d	1(1)		1(1) 1(5) 1(6)	1(6)	1(6)		1(5)			1(1)	
n	1(1)	1(1) 1(5) 1(6)		1(6)	1(6)		1(5)			1(1)	
A		1(6)	1(6)		1(6)						
Lo		1(6)	1(6)	1(6)		1(3)	1(4)	1(3)	1(4)		
Ls					1(3)			1(3)			
Lc		1(5)	1(5)		1(4)				1(4)		
S					1(3)	1(3)				1(2)	1(2)
Sc					1(4)		1(4)				
R	1(1)	1(1)	1(1)					1(2)			1(2)
F								1(2)		1(2)	

Table 8.6: *Initial efforts and influences.*

Parameters		**D**	**d**	**n**	**A**	**Lo**	**Ls**	**Lc**	**s**	**sc**	**R**	**F**
Start	E	3(1)	3(1) 3(6) 2(5)	3(1) 3(6) 2(5)	3(6)	3(6) 2(4) 2(3)	2(3)	2(5) 2(4)	2(2) 2(3)	2(4)	2(2) 3(1)	2(2)
	Inf	3	8	8	3	7	2	4	4	2	5	2

Table 8.7: *Problem instance of spring design example.*

Parameter	***State***
D	unknown
d	unknown
n	unknown
A	unknown
L0	Known
Ls	unknown
Lc	Known
s	unknown
sc	unknown
R	unknown
F	Known

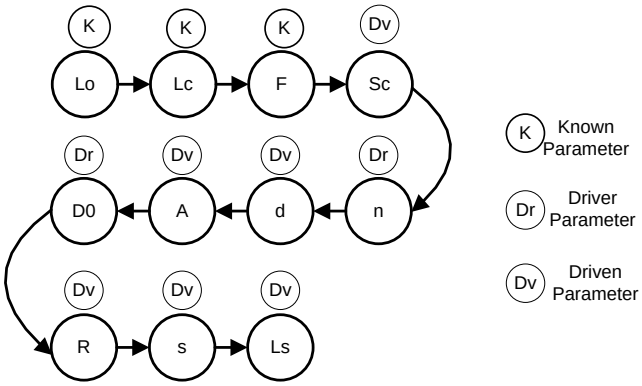


Figure 8.5: *Instantiating order of parameters in spring example.*

Table 8.8: *Results of KGM transformation in example.*

		D	**d**	**n**	**A**	**Lo**	**Ls**	**Lc**	**s**	**sc**	**R**	**F**
Step 1	E	3(1)	3(1) 2(6) 1(5)	3(1) 2(6) 1(5) **	2(6)	0 *	1(3)	0 *	2(2) 1(3)	0(4)	2(2) 3(1)	0 *
	Inf	3	6	6	2	0	1	0	2	0	4	0
Step 2	E	2(1)	2(1) 1(6) 0(5) ***	0	1(6)	0	1(3)	0	1(2) 1(3)	0	1(2) 2(1)	0
	Inf	2	3	0	1		1	0	2	0	3	0
Step 3	E	1(1)	0	0	0(6) ***	0	1(3)	0	1(2) 1(3)	0	1(2) 1(1)	0
	Inf	1	0	0	0	0	1	0	2	0	2	0
Step 4	E	1(1)	0	0	0	0	1(3)	0	1(2) 1(3) *	0	1(2) 1(1)	0
	Inf	1	0	0	0	0	1	0	2	0	2	0
Step 5	E	1(1)	0	0	0	0	0(3) ***	0	0	0	0(2) 1(1) ***	0
1.	Inf	1	0	0	0	0	0	0	0	0	0	0
Step 6	E	0(1) ***	0	0	0	0	0(3) ***	0	0	0	0(2) 1(1) ***	0
	Inf	0	0	0	0	0	0	0	0	0	0	0

State transition: * unknown-known, ** unknown-driver, *** unknown-driven

Part IV

Results and Conclusions

Chapter 9

Integration and Implementation

This Chapter discusses the integration of the methods discussed in Chapters 5, 6, 7 and 8. Two computer implementations demonstrating the resulting method are presented: one generic toolbox to test the proposed methods and one specific implementation that automates CSIM design.

9.1 Introduction

This chapter explains how the methods proposed for complexity management are integrated into a methodology for CDS. The resulting methodology is named *CDS by Complexity Management* (CDS-CM). A preliminary generic software implementation of this methodology has been developed, and is described in Appendix C. The design of a transmission system, presented in Subsection 9.2.4, is used as example to demonstrate it. Section 9.3 presents an specific implementation of this methodology to CSIM design.

9.2 Methodology: CDS-Complexity Management

The flow chart in Figure 9.1 shows how the techniques proposed in this thesis can be integrated to result in a methodology for CDS: CDS by Complexity Management (CDS-CM). The methodology consists of two general phases: initialization and generation. The *initialization* phase consists of transforming the design problem into a formal model of a problem class. The *generation* phase takes over the automatic generation of design solutions.

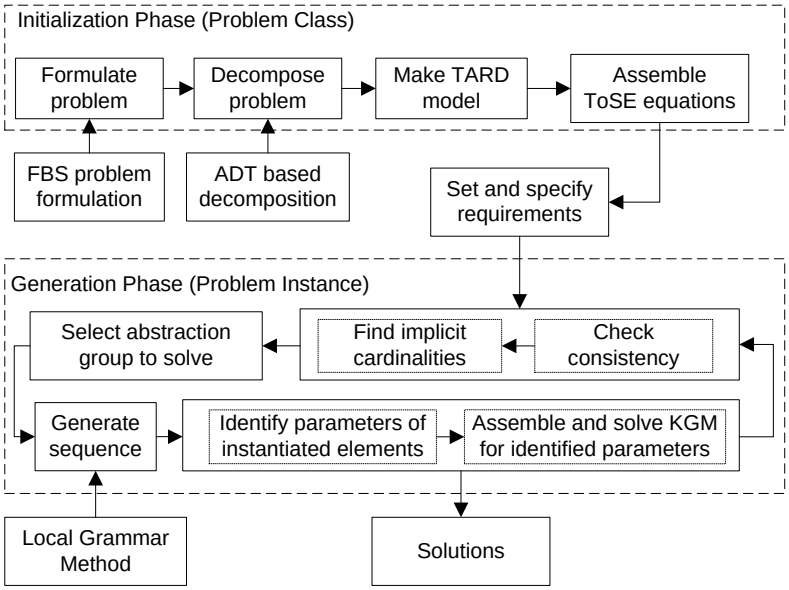


Figure 9.1: *General procedures in CDS by Complexity Management.*

9.2.1 Initialization Phase

The initialization aims at developing formal and consistent design problem models. This is achieved by:

1. Problem formulation: consists in obtaining a consistent model of the information contents of the problem as proposed in Chapter 3. This thesis proposes FBS design formulation as means of performing this task.

2. Problem decomposition: consists in reducing the dimensionality of the problem by decomposing it into smaller problem chunks with lower degrees of complexity. By doing so, search processes in the generation phase can focus on specific levels of abstraction. This thesis proposed ADT problem decomposition as a method for doing so.

3. Making the TARD model: consists of representing a problem class by means of the four types of building blocks in TARD: Elements, C-relations, H-relations and ACO-relations. Once the problem is translated into its TARD form, its formulation can be reused for different problem instances. Furthermore, the model can be integrated into other TARD models to be reused in other design problem formulations.

4. Assembling ToSE equations: Is done as explained in Chapter 7. These

equations allow setting requirements on the problem's topology. During the generation phase, ToSE equations allow controlling the generation of design solutions.

9.2.2 Generation Process

Once the initialization phase is completed, a generation phase determines how to instantiate the elements and relations in the problem. It is important to notice that the method only determines strategies, or orders, in which the problems can be solved. The algorithms for instantiating these building blocks have been kept out of the scope of this research. The generation procedure consists of four activities: (1) checking the consistency and calculating implicit cardinalities, (2) selecting an abstraction-groups, (3) solving that abstraction-group by generating sequences, and (4) solving the values of the parameters contained in each element.

Checking Consistency

After ToSE is derived, the first operation performed is to check for inconsistencies in the cardinality values. A consistent set of equations is required for determining valid solutions. There are two consistency rules that have hold:

1. Balance equations have to meet equality.
2. The value of each cardinality has to be a positive integer.

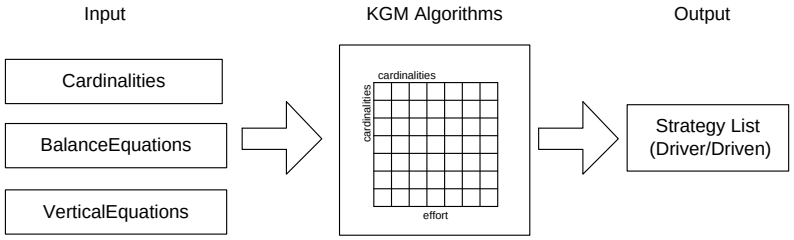
In case that not all cardinalities are known, the ToSE equations are used to determine if any other cardinality can be directly computed. This operation reduces the degree of freedom of the problem.

Identifying Abstraction-group to Solve

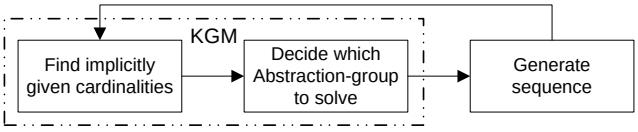
The KGM algorithm described in Chapter 8 is applied to ToSE to identify which abstraction-group to solve, as indicated in Figure 9.2(a). The ToSE equations and the cardinalities values are fed into the KGM algorithm, obtaining a list of driver cardinalities. The cardinality with the lowest effort determines the abstraction-group to solve. As the successive generation of a sequence will fix the cardinalities of other elements and relations, the KGM algorithm has to be applied recursively after obtaining the solution of one abstraction-group, as indicated in Figure 9.2(b).

Generating Sequences

Simple abstraction-groups are solved by attributing values to the elements cardinalities according to the order prescribed by the KGM algorithm. In complex abstraction-groups, sequences are generated using the Local Grammar method described in Section 7.3.



(a) KGM input and output.



(b) Abstraction-group selection algorithm.

Figure 9.2: *Choosing abstraction groups.*

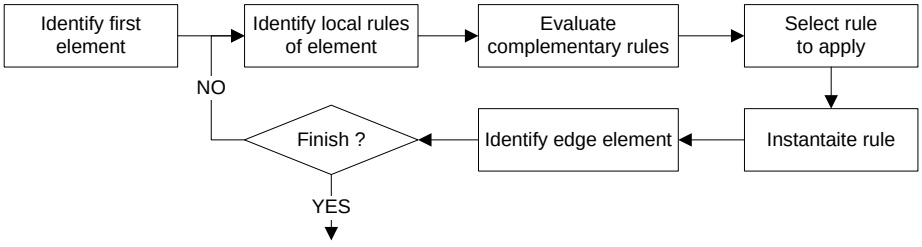


Figure 9.3: *Generation algorithm for local grammar method.*

Figure 9.3 shows a simple algorithm for generating sequences using this method. The local grammar rules are selected by checking the references to outgoing C-relations of the element at hand. Thereafter, it is checked if any complementary rules can be applied. The choice of which of the remaining rules to choose is done by following a certain algorithm, e.g. randomly. If the cardinalities i and j of the chosen C-relation are not known yet, they have to be generated. Generated sequences are stored as a sequence object in the corresponding H-relation instance. The corresponding cardinalities are calculated according to the sequence equations. If the KGM analysis does not result in any more drivers or drivens, the topology of the problem is completely defined and is not subject of further generation.

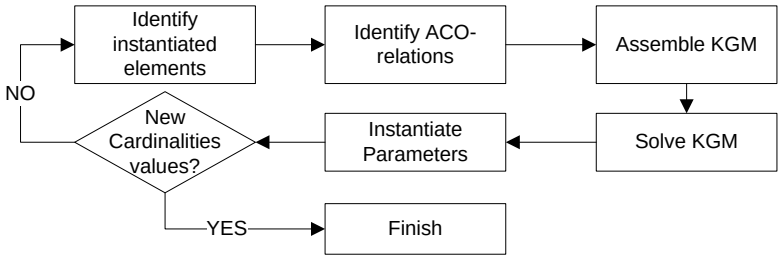


Figure 9.4: *Identifying and solving parameter values.*

Identifying and Solving Parameter Values

Instantiated elements have parameters with no values assigned. These steps consists in identifying which parameters appear as consequence of instantiating elements, and attributing them with values, as indicated in Figure 9.4. In order to do so, the ACO-relations in which the parameters are present are listed too. This information is then used to assemble a KGM and determine the order in which they can be solved. In case the proximity relations (also an ACO-relation) determines the instantiation of an element, the procedure is repeated taking into account the parameters of the instantiated elements.

9.2.3 Generic Implementation

The method described in Section 9.2 has been implemented into a generic software toolbox. The software automatically generates solutions for design problems represented by a TARD model. Appendix C describes the generalities of the implementation of the TARD building blocks and the construction of the ToSE equations.

9.2.4 Example: Drive train Design

This example considers a schematic composition of a vehicle drivetrain, as for example the one shown in Figure 9.5. The goal of this example is to show the collaborative integration of TARD, the Local Grammar method, ToSE and the KGM algorithm by the implemented toolbox. For explanatory reasons, parametric descriptions of the elements have not been taken into account.

Problem class

The TARD model of this example is shown in Figure 9.6. It consists of 16 elements (in addition to the zero-level element) and three abstraction-groups for more detailed levels. The only complex abstraction-group corresponds to $H1$.

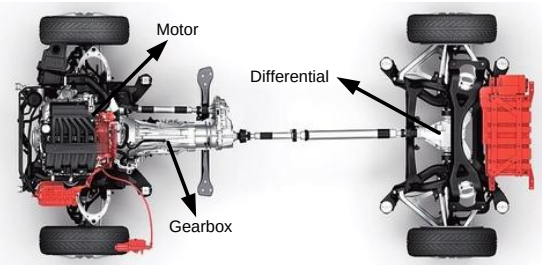


Figure 9.5: *Sketch of a drivetrain in a car.*

Depending on the amount and position of known cardinalities, the solution space and shape changes and expands.

Problem instance

The objective of this problem instance is to generate a topology that does not violate the constrains imposed by the specified cardinalities, which are:

- One motor, connected to one $axle_1$: $e_{Motor} = iC_1 = jC_1 = 1$.
- One input *shaft* and one output *shaft* per *gearbox*: $e_{GearinputShaftlocal} = e_{GearoutputShaftlocal} = 1$.
- Due to their functions, the components of the differential are all connected one-on-one, with the exception of the one-on-two connection between the *planetGear* and *sideGear*: (i, j) $C_{12} = C_{13} = C_{15} = iC_{14} = 1$ and $jC_{14} = 2$
- A two-on-one C-relation for C_3: $iC_3 = 2$ and $jC_3 = 1$.
- Two wheel axes connected to the differential: $iC_5 = 1$ and $jC_5 = 2$.

The last point poses an interesting test for the application: there should be twice as many instances of the element *differential* as of the element $axle2$. The next step is to feed this graphical network together with the defined cardinality values into the application in order to obtain a 1^{st} order representation composed of 1^{st} order elements, C-relations and H-relations object instances. This is achieved by hard coding the data into the application by means of an input class. On the basis of this input, the algorithm is able to derive a solution in the form of a 2^{nd} order network.

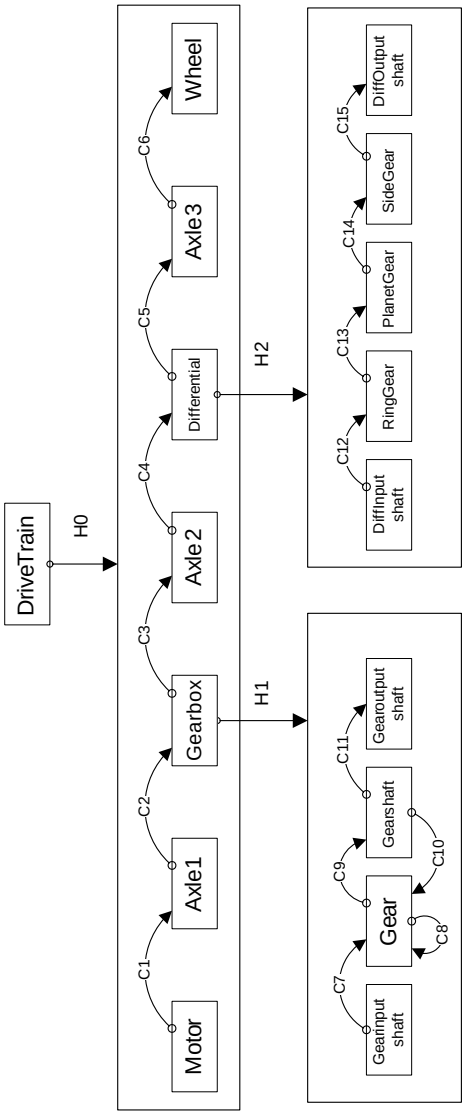


Figure 9.6: *TARD representation of drive train example.*

Problem solution

Figure 9.7 shows one of the automatically generated solutions for the problem instance descrived before. This solution demonstrates the functionality of the implementation, especially in the treatment of the system of equations. The 2^{nd} order network resulting from the determination of the cardinalities represents the topological solution for a drivetrain design. Here, the names of the element instances are followed by an index starting at zero, which serves as ID for identifying different instances of the same element class. For example, $gearbox_{00}$ and $gerbox_{01}$ resemble the first and the second instance of the type *gearbox*. The solid arrows represent C-relation instances connecting the element instances. A C-relation connecting one to two elements and vice versa is represented by two separate arrows, as for example between elements *Axle1* and *Gearbox*. Accordingly, the dotted curved arrows indicate the H-relation instances, which are pointing towards the C-relation instances in the respective abstraction-groups.

The design solution generated by the method consists of a single motor connected to two gearboxes via a single axle. There are two gearboxes connected to a single axle, which in turn drives two differentials. Finally, the two axles behind each differential are connected to a single wheel. It is worth mentioning that the multiple existence of the abstract elements *gearbox* and *differential* have resulted in two independent instances of their abstraction-groups. Proof of a correct execution is the existence of twice as much gearbox instances as *Axle2* instances. As the network is consistently connected, it is demonstrated that all unspecified cardinalities are properly calculated or generated according to the theory.

The generated solution (single motor, double gearbox and double differential) is analogous to the drivetrain of four-wheel drive vehicles having a normal and low range gearbox with differential at each axle.

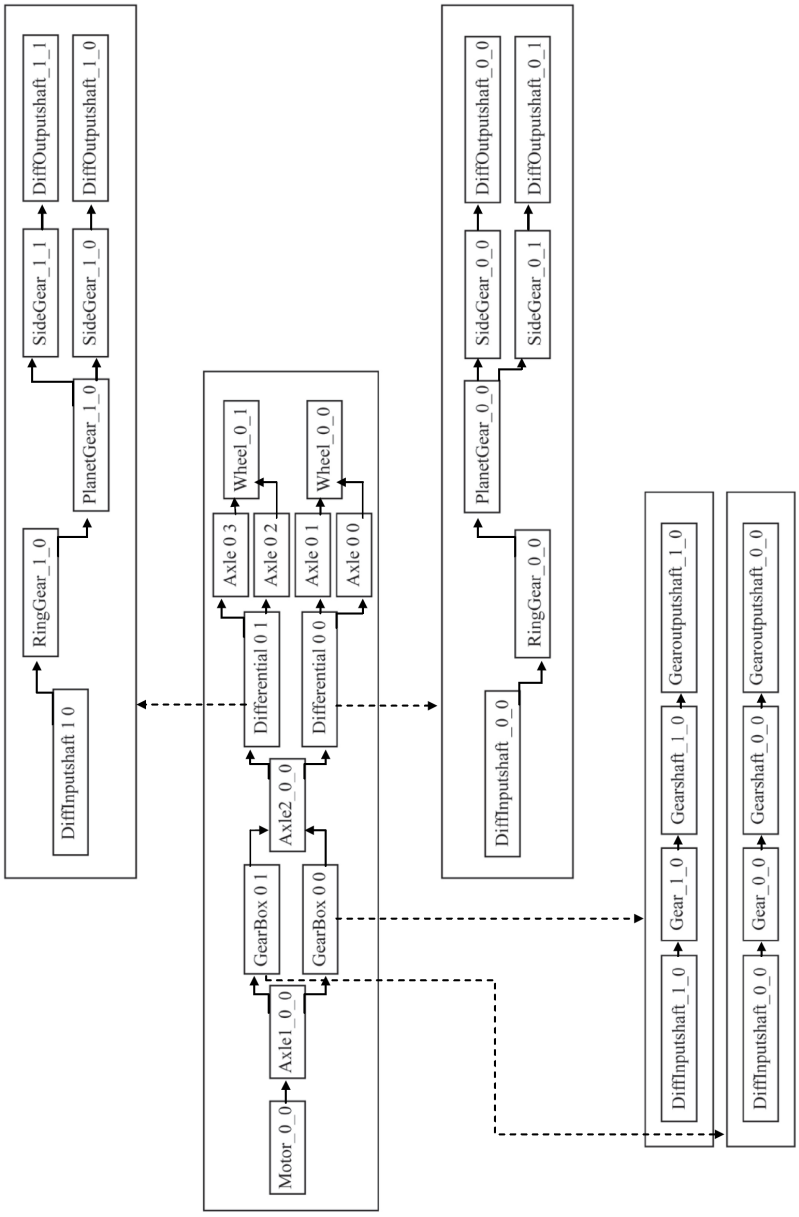


Figure 9.7: *The resulting 2^{nd} order instantiated network representing a solution of the generation process (generated automatically by the implementation.)*

9.3 Automating CSIM Design

Applying the complexity management techniques to CSIM design has resulted in a a bottom-up recognition approach (see Subsection 7.3.4). The method and algorithms for cooling generation have been implemented using C#© [55]. SolidWorks© [14] is used as interface for modeling the 3D mold parts. Furthermore, a User Interface (UI) was developed using SolidWorks© API [14]. This section presents the automation method from a general point of view, and the results from its implementation into a software tool.

The first and second steps of the initialization phase have been performed in the examples shown in Chapter 5. This result is used to build the TARD model presented in Figure 9.8. The considerations taken for making the TARD model and the ToSE equations that follow from this model are presented in Appendix B.

9.3.1 Synthesis Strategy

Figure 9.9 presents a summary of the steps in the CSIM design automation strategy that follow from the TARD model and ToSE equations. The method consists of:

1. Making a voxel model of the solid parts *Mold Part* and *Plastic Parts*. This step consists in transforming the requirements from the shape model into fields, as indicated in Chapter 5, Subsection 5.3.3.

2. The second step consists in generating a mesh of *Points* on top of the voxel model and attributing color to the points according to the relations shown in Table 5.3. By doing so, a pool of elements *Points* is obtained. As consequence, the abstraction-groups *absorberChannels*, *connectorChannel*, *inputSegment* and *outputSegments*, are generated by using a recognition approach, as indicated in Subsection 7.3.4.

3. Generating *absorberChannels* by connecting points using the Local Grammar method and a recognition approach.

4. Generating cooling circuits by connecting *Absorber channels* with *Connector channels*. The later are generated by assembling pairs of green points also using the Local Grammar method and a recognition approach. Furthermore, by connecting *Absorber channels* and *Connector channels* to *inputSegment* and *outputSegments* fully defined circuits are obtained.

5. Generating cooling systems is performed by assembling combinations of the previously generated cooling circuits. Here again, all obtained circuits form a pool of elements from where a recognition approach determines possible cooling system configurations.

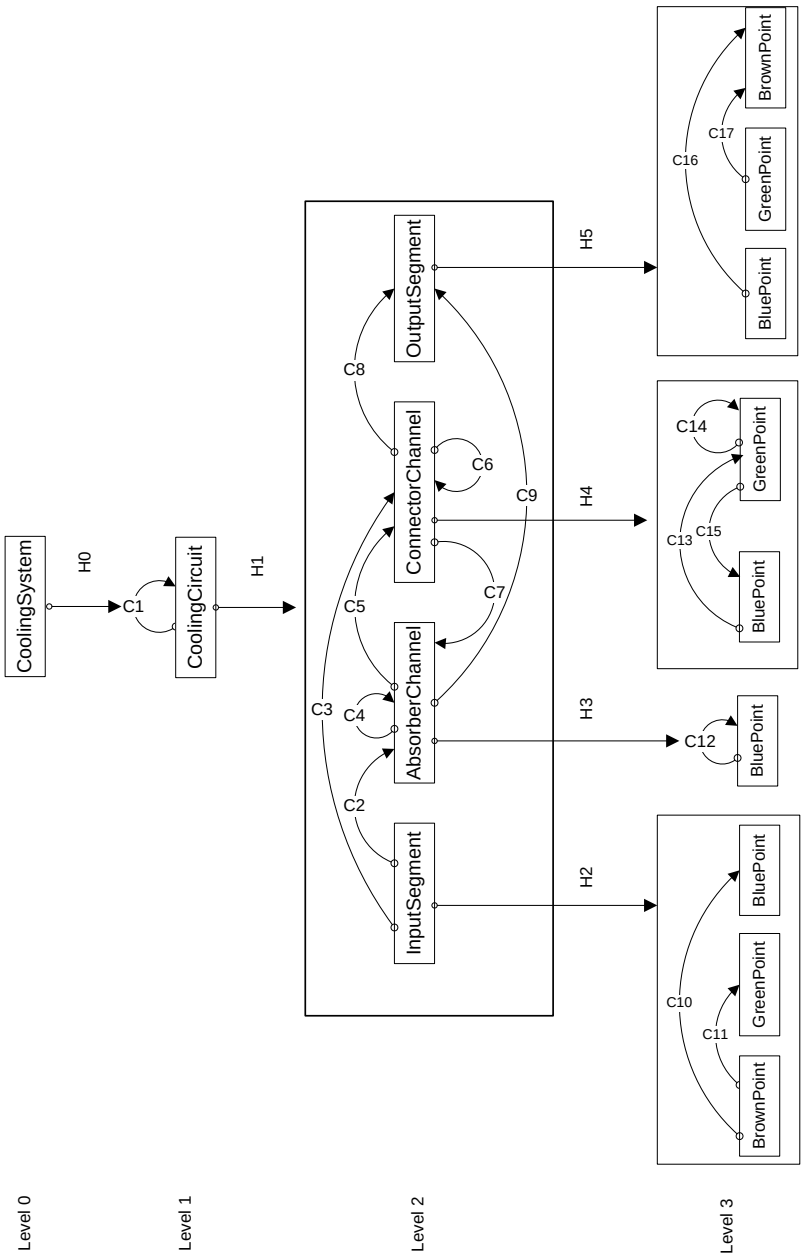


Figure 9.8: *TARD model of CSIM problem.*

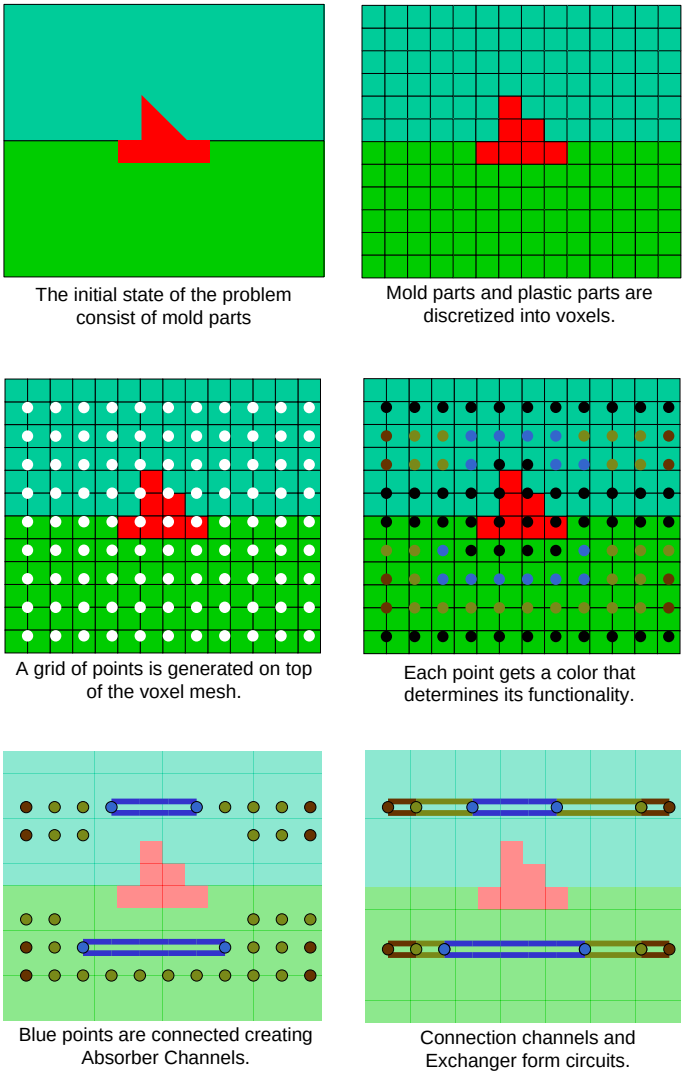


Figure 9.9: *Method for automating CSIM design.*

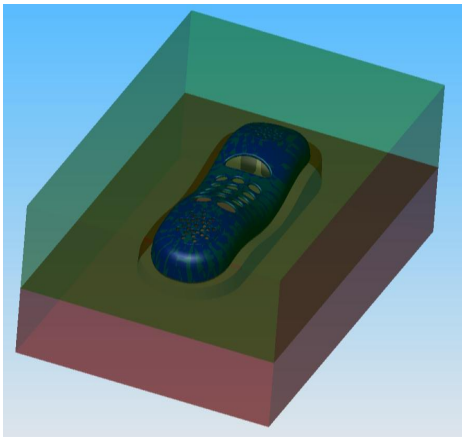


Figure 9.10: *Mold of telephone used as example.*

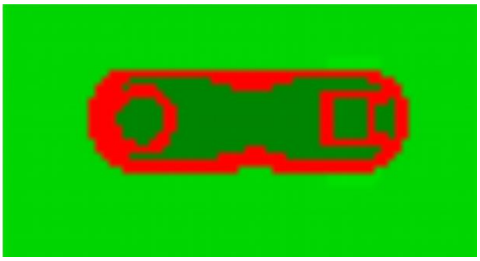
Table 9.1: *List of attributes of a voxel element.*

Attributes	***Description***	***Model***
Core	Identifies the core of the mold	Boolean
Cavity	Identifies the cavity of the mold	Boolean
Product	Identifies the product or plastic part of the mold	Boolean
Inlet/outlet	Identifies if a surface is used for inlets or outlets	Boolean
Non drillable	Identifies if a surfaces cannot be drilled	Boolean
Size	Defines the size of the vertices of the voxel	Boolean
Position	Defines the position of the voxel	double [i,j,k]

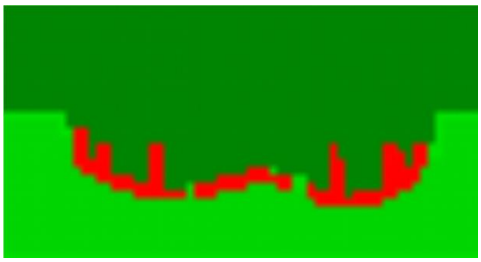
The ToSE (shown in appendix B) equations are used to keep topological consistency of the resulting solutions. The following sections explain each of the step of the method at the hand of telephone mold example shown in Figure 9.10.

Voxel Mesh Generation

Creating a mesh of voxels is the first step of this method. By using voxels, the geometric model can be transformed into a logic one. The idea of using voxels resulted from applying the ADT Based Decomposition to the physical domain. Although this transformation results in a loss of geometric information, it also eases solving spatial constraints. Table 9.1 summarizes the attributes of a voxel object. Figure 9.11 shows two sections of the voxel mesh of the telephone mold shown in Figure 9.10. Here, the core is indicated in dark green, the cavity in light green and the part in red.



Horizontal section



Vertical section

Figure 9.11: *Sections of the telephone voxel mesh model.*

Points Generation

After making the voxel mesh, a 3D grid of points is generated. This grid is set inside the voxel mesh model of the mold. The distance between points in the grid is based on relation PCC-2, whos values can be found in [28]. The objective of this step is to define the spatial locations in the mold where cooling channels can eventually be placed as well as those places where it cannot. To do so, a "color" is attributed to each point in the grid. This assigned attribute is used to characterize the function of the point. These are:

- Blue: defines points that can be used to create an Absorber channel.
- Green: defines points that can be used to create a Connector channels.
- Brown: defines points on the surface of a core and cavity where Exchanger channels can be placed.
- Grey points: define points nearby the melt where channels cannot be placed to overcome mold break.
- Black points: define points where channels cannot be placed to avoid mold break.

Table 9.2: *Logic relations defining the color of points.*

Point color	***Surrounded by***
Green	(core voxels) or (cavity voxels)
Brown	[(core voxels) or (cavity voxels)] and (Exchanger voxels)
Black	[(core voxels) or (cavity voxels)] and [(non drillable voxels) or (product voxels)]
Grey	[(core voxels) or (cavity voxels)] and (product voxels)
Blue	[(core voxels) or (cavity voxels)] and [Grey points]

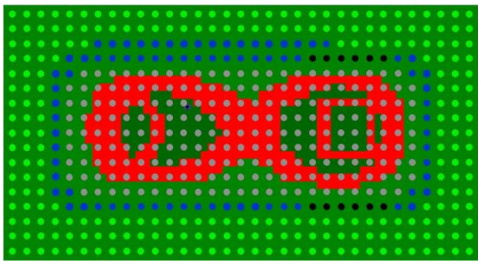
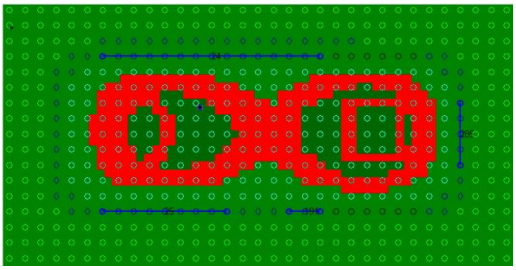


Figure 9.12: *Section of telephone mold with points.*

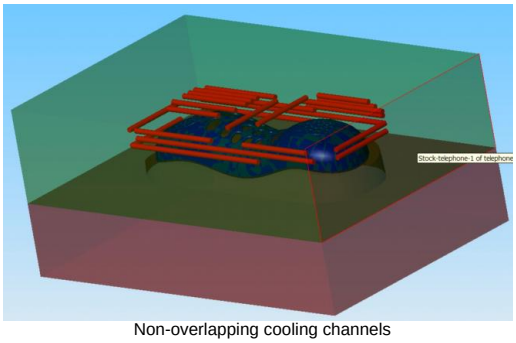
The color of a point is determined as a function of the voxels surrounding that point. In Table 9.2 the logic relations to determine the colors are presented. This design step allows determining the primitive element Point for all three types of channels. Furthermore, this step determines the solution space of each design function in the problem, namely, to cool the plastic part (blue points), to transport the coolant (green points) and to exchange the coolant with exterior heat dissipation devices (brown points). Figure 9.12 shows the result of applying the point grid to the case of the telephone mold.

Absorber Channels Generation

Once all the points have been generated, the abstraction-group corresponding to the element *absorberChannel* is solved by using a recognition strategy. This is done by recognizing patterns of blue points that can be connected to form a channel. The criterion used in the current implementation of the method is based on: drilling manufacturing technique, minimum channel length, minimum heat absorption index and minimum average homogeneity index. An example of channels for the example of the telephone is presented in Figure 9.13.



Section showing cooling channels



Non-overlapping cooling channels

Figure 9.13: *Group of aleatory selected absorber channels in the telephone mold.*

Circuits Generation

Cooling circuits are designed by combining *absorberChannels* capable of forming a circuit. This is done by defining sequences of C-relations in the abstraction corresponding to the element circuit. A recognition strategy is used, following the indications presented in Chapter 7. Knowledge about the construction of feasable sequences are based on knowledge rules of expert designers at Philips ATC. The A* algorithm [36] is used as recognition algorithm that determines paths of *connectorChannels* between *absorberChannels*. The resulting path determines the layout of the *connectorChannels*. Furthermore, *inputSegements* and *outputSegements* are generated by recognizing which *brownPoints* are closer to the obtained circuits.

Cooling Systems Generation

Cooling systems are generated by recognizing combinations or *coolingCircuits* satisfying the physical coherence constrains of the problem. In the implementation developed in this work, this is performed by randomly combining *coolingCircuis* and assessing if they violate any of the constraints.

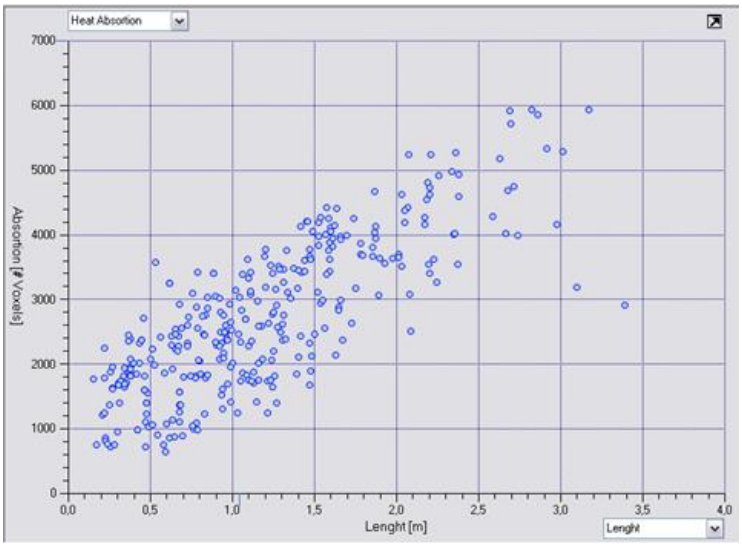


Figure 9.14: *Solution space of CSIM design for telephone mold.*

9.3.2 Results

Following these steps results in several candidate cooling systems. Figure 9.14 shows the solution space for the case of the mold in Figure 9.10. Each point in the graph corresponds to one cooling system layout. The y-coordinate corresponds to the performance parameter heat absorption, while the x-coordinate corresponds to the length of the system. As the figure shows, a wide cloud of solutions is obtained for this example. By assessing their performances, desired solutions can be selected. Figure 9.15 shows four cooling solutions drawn in SolidWorks©. The results suggest that this method is capable of generating feasible cooling solutions.

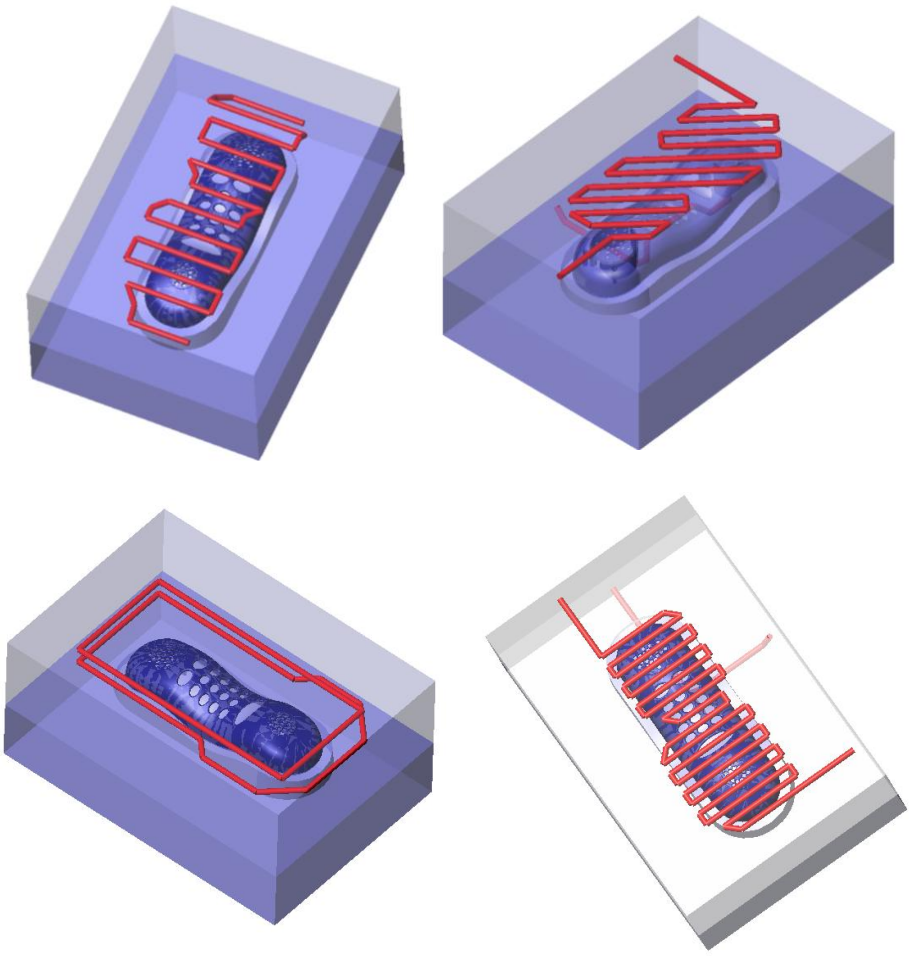


Figure 9.15: *CSIM design solutions for telephone mold.*

Chapter 10

Conclusions & Recommendations

This chapter presents a discussion on the usage of complexity management in the development of methods for computational design synthesis. The main issue is whether such techniques can support the generic development of Computer Aided Synthesis tools. The chapter finishes presenting recommendations for further research.

10.1 Conclusions

In general terms, automating routine design problems can be performed in two different fashions: developing specific methods for specific problems, or developing generic methods for problem families. It is the second approach which can enable the future development of CAS systems, and the one this thesis dealt with. Such an approach is confronted with three main challenges. Firstly, design problems have to be translated into terms that allow their study independent of its semantic contents, thus to define a common language. Secondly, basic problem characteristics have to be identified, which consists in finding the common ground of these problems. And thirdly, general problem solving approaches have to be developed. This thesis contributed to solving these challenges by investigating complexity in design. Its result is a generic approach for automating artifactual routine design problems. Its computer implementation demonstrates the method can be used for automating such problems. However, further research is required to achieve a true generic software for design automation, as it is explained in Section 10.2. The following sections present a discussion on these issues.

First challenge: a common language for routine design

The first challenge has been solved by developing a scheme for formulating artifactual routine design problems, described in Chapter 3. The scheme has been developed by mapping models of information types found in design literature to schemes for formulating problems found in problem solving theory. The information contents has been further characterized by classifying commonly used mathematical models in design automation in the following groups: parametric, spatial, shapes, fields and topology. This has enabled benchmarking design problems that at first sight seem very different. The results represent a common language for understanding the similarities and differences of between routine design problems. Furthermore, the solution approach to one given problem can be extrapolated to others, as long as both formulations are equal.

Second challenge: a generic structure for design problem

By making an analogy to the structure of analysis problems (which deal with understanding the behavior of "things"), this thesis presents a new framework for structuring design problems. The structure consists of three levels of abstraction: the problem class, the problem instance, and the problem solution. A problem class is formed by non instantiated elements, parameters and relations. A problem instance has partially instantiated elements, parameters and relations, which represent the requirements of the problem. Problem solutions are formed by fully instantiated elements, parameters and relations. This structure allows generalizing design problems by:

- defining the types of representations required for modeling each abstraction,
- studying the complexity of each abstraction by analyzing the configuration of the used representations,
- developing procedures and operations to solve each problem domain, and
- identifying common problem structures.

This framework, described in Chapter 4, constitutes one of the main contributions of this work, as it allows studying design problems in abstract terms and identifying the characteristics of design problems from where complexity arises.

Mapping the structuring framework to ADT model of complexity, has resulted a new model of design complexity for routine design. It has been found that the complexity of problem classes is related to the consistency of the information contents and the problem formulation. One cause of complexity results from having a less suitable problem formulation. Another is the lack of differentiation between problem chunks and its interrelations. In this sense, in complex problem classes the design problem is not addressed correctly and it has a high dimensionality (few levels of abstraction).

Complexity in problem instances depends on:

1. The distribution of the requirements: concentrated in the top abstraction layers, in the bottom layers or a mix of both.

2. The constrains imposed to the cardinalities of elements. Cardinality is a variable that defines the number of times an element can be instantiated. If cardinalities are unknown and unbounded, the number of instantiated elements (and therefore the number of parameters) explodes.

As these complexities can be identified independently of problem specific semantics, they represent a generic approach for determining what the problem to be solved is. This brings the issues of solving routine design problems to that of managing their complexity, which verifies the first hypothesis of this research.

Third challenge: generic solving approaches

Different complexity management methods have been developed. Figure 10.1 presents an overview of these methods.

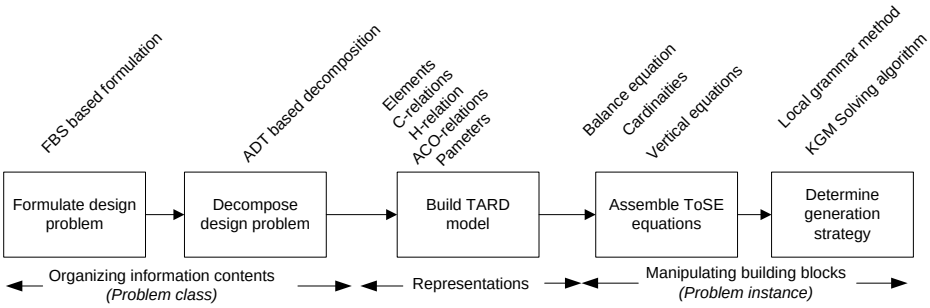


Figure 10.1: *Integrated approach to complexity management.*

Information Contents

A first group consists of two methods that aim at organizing the information contents. One method targets the development of consistent problem formulations, while the second focuses on decomposing a problem into different levels of abstraction. Both methods target complexity of problem classes. By using the FBS model, the first method is capable of keeping a detailed track of the relations among the information contents of the problem. The second method is founded upon ADT's axioms ruling good designs, which sets a formal criteria for decomposing design problems. The application of these methods enable the initialization of a synthesis process, which according to Cagan et al. [4] has not received enough attention in literature. The result of applying these two methods is a well organized design problem with low dimensionality.

Representations

The second group consists of a new framework for the computational representation of design problems named TARD. TARD integrates four types of building blocks (elements, C-relations, H-relations and ACO-relations) to support mathematical models for design automation by exhibiting the characteristics of the design formulation scheme and problem structure previously described. In TARD, elements group parameters, ACO-relations determine relations between parameters, C-relation describe how different elements types can be connected, and H-relations describe the relations between different levels of abstraction. This configuration has number of advantages:

1. allows developing solving strategy methods as function of the organization of building blocks,
2. abstraction groups can be solved independent from others,
3. several domains can easily be represented (juts adding extra H-relations),
4. it enables both bottom-up and top-down design strategies, and
5. solutions can be generated by using either parametric or grammar approaches.

Building Blocks Manipulation

The third group of methods developed in this research relates the structure of a problem instance to a strategy for solving it. This part of the research focused on parametric and topology models. The result is a theory about the parametrization of topologies represented in TARD, and two methods for manipulating building blocks. One method deals with elements, while the second does it with parameters.

ToSE, or the **To**pology **S**ystems of **E**quations, represents topologies in the form of algebraic equations. These equations have three specific functions: keep consistency of the topology, identify distributions of requirements, and control the instantiation of elements. Furthermore, ToSE equations are assembled for a given TARD model following strict rules, which allows for a straightforward computer implementation. The main advantage of ToSE is that topologies can be solved using known constraint solving methods.

This thesis developed the Local Grammar method as means of solving TARD instance problems with under constrained ToSE equations. This method is an adaptation of the grammar approaches described in Chapter 2. However, it differentiates from them in two aspects:

- the generation of solutions is controlled by the cardinalities in ToSE, and
- it integrates functions that determine if the generation algorithms should exhibit creation or recognition characteristics.

Creation corresponds to a top-down approach, for example, assessing first the ***types*** of elements that can be connected, and then instantiating and connecting them. Recognition, on the other hand, corresponds to a bottom-up approach. For example, a pool of ***instantiated*** elements exists, and its characteristics determine how they can be connected. This characteristic, and the fact that it is generic to problems represented by TARD, opens a new way of solving design problems. For example, assessing the differences in the design solutions of problems that are specified in terms of *what we want* (requirements at the top abstraction levels) vs. those specified in terms of *what we have* (requirements set at the bottom levels of abstraction). This moves the task of designers from *finding solutions* towards *designing the right problems*.

The KGM solving algorithm is proposed for determining the order in which parameters can be instantiated. The algorithm gets its name from the Knowledge Graph Matrix used for representing the structure of parameters and relations. The algorithm is based on two complexity measures that can be calculated for each parameter, namely, the effort and the influence. The effort relates to the number of parameters that have to be known in order to solve another parameter. The influence is related to the number of parameters that can be solved if a given parameter is known. The algorithm finds a solving order by assessing which parameters have low efforts and high influences. The algorithm has demonstrated to be effective in underconstrained problems.

By integrating all these methods, a new Computational Design Synthesis methodology is presented: CDS by Complexity Management. The methodology is demonstrated by two implementations. One is the case study of this research: design of cooling systems for injection molding (termed CSIM design). The second is a preliminary generic implementation. The latter permits automating a design problem by entering the TADD model of a problem instance.

Final Remarks

From a general point of view, two types of methods are required for automating design problems:

- Solving strategy: focus on determining road maps indicating the order in which elements and parameters are instantiated.
- Instantiating elements and parameters: regard instantiation of elements and parameters such that the goals of the design are meet. Examples are random generation, case based selection, optimization techniques, etc.

Furthermore, automating methods can also be classified into *static* or *dynamic*. Static methods consist in solving first the strategy, and latter the instantiation. For example, evolutionary methods (described in Chapter 2) consist of a fixed strategy, having a random instantiation of parameters. Dynamic methods determine the strategy as the elements and parameters of the problem are being

instantiated, thus, in an integrated fashion. For example, A-based design (described in Chapter 2) is a dynamic method, as both the solving strategy and instantiation of elements and parameters occurs as the problem is being solved.

Using these classifications, the work on complexity management presented in this thesis focused on determining solving strategies using a static approach. The main contribution is that it provides a generic approach for automating artifactual design problem. The integration of these methods has the advantage that it permits the reuse of existing problem formulation, the use of standard solving methods, and the development of computer based design tools. Furthermore, it has been demonstrated that these techniques can be used in a collaborative fashion for automating the generation of solutions by managing design complexities, corroborating the third hypothesis driving the research presented in this thesis.

10.2 Recommendations

Two main lines of further research have been identified: one is towards the development of CAS, and another deals with the automation of CSIM design. In this section, the recommendations firstly focus on the development of generic methods for design automation, and secondly on transforming the existing CSIM designer prototype into a tool for standard usage.

Towards CAS

In order to achieve full automation of artifactual routine design, it is required to research the complexity of problems that are represented in the domain of *fields*, *spatial* and *shapes*. A possible approach for doing so consists in collecting problems with such characteristics and assembling its corresponding TARD models. As TARD is based on an ontology that considers these domains of representations, it can now also be used for understanding its complexity characteristics. There is also a need to understand the relations between algorithms for instantiating elements/parameters and the domains of design representations.

From the point of view of developing general propose CAS systems, the following road map is proposed:

1. At present, TARD models are implemented in programming time. A user interface for implementing TARD models would lower the human effort required for implementing new problems. The resulting software would serve as experimental setup for testing new methods and algorithms.

2. Integrate functional representations into TARD to enable high level abstraction reasoning for engineering design problems.

3. Develop a software architecture that permits plugging-in new algorithms for both solving strategies and instantiating elements and parameters.

4. Explore the possibility of using TARD to represent functions and behaviors. This could be used as framework for researching methods for automating innovative and creative designs. This would also enable Internet based searches for components, constraints, and analysis relations.

This road map represents an exiting field of further research, with a possible large impact in current design practice.

CSIM Design

The current implementation of CSIM design generates solutions that cover a large portion of the solution space. However, many of the generated solutions are not realistic in terms of its applicability. This problem has three causes. One is the lack of constrains imposed to the instantiation process. Another is the types of instantiating algorithm used for the automatic generation. The third cause is that the analysis of solutions is based on a qualitative method. It is recommended to tackle these problems by:

1. Introducing new pattern based rules by further assessing the knowledge used by expert designers.

2. Implementing other instantiating algorithms, as for example Genetic Algorithms. The idea is to let GA identify combinations of absorber channels that maximizes the heat abortion of the cooling system while minimizing temperature differences in the part.

3. Developing a new analysis method. At the moment this thesis is being written, an analysis method based on a multigrid approach is being developed to solve this problem.

By researching these issues, a new tool can be obtained that fully automates the design of CSIM.

Acknowledgments

In the last four years, I think the most difficult question somebody could make me was *"and tell me, what are you doing?"* I never knew how to answer it correctly. On the one hand, if I would say *"I am doing a PhD"*, people would respond *"so, you are still studying"*. On the other hand, if I would answer *"I work as researcher"*, people would say *"doing what?"*, and I would respond *"doing a PhD"*, getting again the answer *"ah, so..., you are still studying"*.

Knowing I am about to finish my PhD made me feel relived I will finally stop being a *"student"*. One less in the list of difficult questions to answer. However, I have been confronted now with an even more difficult question: "how can I acknowledge all the people that have given me their support during these last four years in one or two pages?" A fair answer would be saying that this is not possible. However, now that I will officially become a researcher, I cannot say this without having given it a try.

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, thanks for the nice discussions and for helping me understand the world of design from a more general perspective. Thank you also for your confidence in my research and in me.

Professor Tomiyama, in one of our first meetings you described my research as translating design problems *from how much* statements *to how many* ones. As the title of this thesis suggests, I still believe this is the best way to describe it. Thanks you for sharing your insights in the field of design research.

Begeleidingscommissie, thank you for showing me the realities of the design processes at industry.

Mi Linda (Diru), siempre me has apoyado y motivado a cumplir mis sueños. Gracias por querer entender siempre que es lo que estaba investigando. Gracias por mostrarme otras perspectivas, y hacerme preguntas difíciles. Gracias por

acompañarme en las noches de trasnocho. Por eso, y mucho mas, este nuevo logro es tan tuyo como mío. Te agradezco todo el amor y la paciencia que me has brindado en nuestros años juntos. Gracias.

Papa y mama, ustedes han sido el mejor ejemplo de que con principios, trabajo, esfuerzo y motivación se es capaz de alcanzar cualquier meta. Les agradezco el siempre haber estado con nosotros, sin necesariamente estar físicamente . A ustedes también, muchísimas gracias.

Hermanos, Maite y Rudy, gracias a ambos por estar siempre pendientes de su hermano. A pesar de la distancia no hemos dejado de compartir muchos momentos bonitos.

Opa, Oma, grote Rudy, Monique, Marion, Ron, Nahidu en Demi, bedankt voor alle steun en gezelligheid in de afgelopen jaren. Het is heel fijn om ook familie in Nederland te hebben.

Sra. Siran y Sr. Victor, les agradezco mucho la confianza que siempre han depositado en mí.

Gracias a todos mis tíos y primos en Venezuela.

I would like to personally thank my direct tutor, Hans. Thank you for guiding me during this academic journey. Your pragmatism really helped me developing this research. Furthermore, the Friday's afternoon wine was a good source of inspiration and motivation.

Fokke, I really enjoyed discussing with you my ideas. I enjoyed even more getting your feedback.

Frans, Georg and Sipke, thank you for showing me the power of exchanging knowledge and techniques from one field to another to find easy ways to solve what might seem as a difficult problem. You were good Friday's afternoon wine companions.

Inge and Ans, thank you for your guidance and support. Thanks you also for showing me the insights of the UT, CTW and OPM organizations.

Two important people I also have and want to thank are Wouter and Wessel. Thank you for the nice discussions, your jokes, and most importantly, thank you for offering me your friendship.

Maarten, Matteijs and Valentina, I really enjoyed working with you in this project. Robert, thanks for always being creative. Boris, thanks for all the nice smoking moments and chats. Thanks to all the department's colleges for the nice chats and cakes. Jorge y Nestor, gracias por las largas horas de amenas discusiones, y por mantener nuestra amistad. Martijn, thanks for offering me your friendship in these years. A los venezolanos en Holanda, Mariana, Ivan y Ana Maria, gracias por todos los momentos cheveres. Gert Jan, Marco, Klass Jan and Stefan, thank you taking part in my research project and for delivering such good results.

I could continue this list for pages and pages, but unfortunately I have to put now an end to it. However, to finish, I want to thank all the people who once asked me what was that what I was doing. Searching constantly for an answer helped me understand what is that what I have done.

List of References

[1] J. Allison, M. Kokkolaras, M. Zawislak, and P. Y. Papalambros. On the use of analytical target cascading and collaborative optimization for complex system design, 2005.

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

[3] L. B. Archer. The need for design education. *Royal College of Art*, 1973.

[4] J. Cagan, M. I. Campbell, S. Finger, and T. Tomiyama. A framework for computational design synthesis: Model and applications. *Journal of Computing and Information Science in Engineering*, 5(3):171–181, 2005.

[5] J. Cagan, I. E. Grossmann, and Hooker. A conceptual framework for combining artificial intelligence and optimization in engineering design. *Research in Engineering Design*, 9:20–34, 1997.

[6] M. I. Campbell, J. Cagan, and K. Kotovsky. Agent-based synthesis of electromechanical design configurations. *Journal of Mechanical Design*, 122(61), 2000.

[7] 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.

[8] M. I. Campbell and R. Rai. A generalization of computational synthesis methods in engineering design. *AAAI Spring Symposium Series*, 2003.

[9] A. Chakrabarti. *Engineering Design Synthesis: Understanding, Approaches and Tools*. Springer Verlag, London, 2002.

[10] W. M. Chan, L. Yan, W. Xiang, and B. T. Cheok. A 3d cad knowledge-based assisted injection mould design system. *The International Journal of Advanced Manufacturing Technology*, 22(5):387–395, 2003.

[11] B. Chandrasekaran, A. Goel, and Y. Iwasaki. Functional representation as design rationale. *IEEE Compute*, 26:48–56, 1993.

[12] L. D. Clive. *Engineering design: a synthesis of views*. Cambridge University Press, 1995.

[13] A. G. Cohn and S. M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundam. Inf.*, 46(1-2):1–29, 2001.

[14] SolidWorks Corporation. *SolidWorks 2007 API Documentation*. SolidWorks Corporation, 2007.

[15] M. B. Douglas. *Plastic Injection Molding: Manufacturing Startup And Management*, volume 4. Sme, 1999.

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

[17] J. Duarte. *Customizing mass housing: a discursive grammar for Siza's Malagueira houses*. Massachusetts Institute of Technology, 2000.

[18] Z. Fan, J. Wang, K. Seo, J. Hu, R. Rosenberg, J. Terpenny, and E. Goodman. Automating the hierarchical synthesis of mems using evolutionary approaches. *Evolvable Machines*, pages 129–149, 2005.

[19] J. S. Gero and U. Kannengiesser. The situated function-behaviour-structure framework. *Design Studies*, 25:373–391, 2004.

[20] J. Gips. Computer implementation of shape grammars. *NSF/MIT Workshop on Shape Computation*, 1999.

[21] V. Goel and P. Pirolli. The structure of design problem spaces. *Cognitive Science: A Multidisciplinary Journal*, 16(3):395 – 429, 1992.

[22] J. Greeno. Natures of problem-solving ability. *Handbook of learning and cognitive processes*, 5:239–270, 1978.

[23] Weiss M.P. Hari A. Icdm an integrated methodology for the conceptual design of new systems. In *Systems Engineering / Test and Evaluation Conference*, 2004.

[24] A. Hatchuel and B. Weil. C-k design theory: An advanced formulation. *Research in Engineering Design*, 19(4):181–192, 2009.

[25] V. Hubka and E. W. Eder. *Theory of Technical Systems: A Total Concept Theory for Engineering Design.* Springer, 1988.

[26] A. Jaklic, A. Leonardis, and F. Solina. Segmentation and recovery of superquadrics series. *Computational Imaging and Vision*, 20:12–39, 2001.

[27] J. M. Jauregui-Becker, H. Tragter, and F.J.A.M. van Houten. Structuring and modeling routine design problems for computational synthesis development. In *CIRP Design Conference on Design Synthesis 2008*, 2008.

[28] J. M. Jauregui-Becker, H. Tragter, and F.J.A.M. van Houten. Toward a bottom-up approach to automate the design of cooling systems for injection molding. In *CAD09*, Reno, Nevada, USA, 2009.

[29] J. M. Jauregui-Becker, W. W. Wits, and F.J.A.M van Houten. Reducing design complexity of multidisciplinary domain integrated products: a case study. In *41st CIRP Conference on Manufacturing Systems*, volume 1, pages 149–154, 2008.

[30] C. Jermann, G.Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: a survey. *Int. J. Comput. Geometry Appl.*, 16(5-6):379–414, 2006.

[31] J. H Johnson. Multidimensional networks in the science of the design of complex systems. In *ECCS 2005 Satellite Workshop: Embracing Complexity in Design*, pages 33–48, 2005.

[32] J. R. Koza, M. J. Streeter, and M. A. Keane. Automated synthesis by means of genetic programming of human-competitive designs employing reuse, hierarchies, modularities, development, and parameterized topologies. In *The 2003 AAAI Spring Symposioum: Computational Synthesis: From Basic Building Blocks to High Level Functionality*, 2003.

[33] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI magazine*, 13(1):32–44, 1992.

[34] T. Kurtoglu and M. I. Campbell. A graph grammar based framework for automated concept generation. In *The International Design Conference 2006*, pages 61–68, Paris, 2006.

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

[36] H. Y. Lee and T. H. Cho. A* - based layout design for gratings allocation. *Computer-Aided Design*, 40(4):455–464, 2008.

[37] C. L. Li and C. G. Li. Manufacturable and functional layout design of cooling system for plastic injection mould. In *International conference on manufacturing automation*, pages 47–54, 2004.

[38] C. L. Li and C. G. Li. Plastic injection mould cooling system design by the configuration space method. *Computer-Aided Design*, 40(3):334–349., 2008.

[39] C. L. Li, C. G. Li, and A. C. K. Mok. Automated layout design of plastic mold cooling system. *Computer Aided Design*, 37:645–662, 2005.

[40] CoreTech System Co. Ltd. Moldex3d/solid-rim reference manual,, 2003.

[41] A. Manish and C. Jonathan. On the use of shape grammars as expert systems for geometry-based engineering design. *Artifitial Intelligence in Engineering Design and Manufacturing*, 14(5):431–439, 2000.

[42] J. P. McCormack, J. Cagan, and C. M. Vogel. Speaking the buick language: capturing, understanding, and exploring brand identity with shape grammars. *Design Studies*, 25(1):1–29, 2004.

[43] Moka consortium Melody Stokes. *Managing Engineering Knowledge: MOKA*. Professional Engineering Publication, 2001.

[44] M. Menges. *How to make injection molds*. Hanser, 1986.

[45] S. Minderhoud and P. Fraser. Shifting paradigms of product development in fast and dynamic markets. *Reliability Engineering & System Safety*, 88(2):127–135, 2005.

[46] C. K. Mok, K. S. Chin, and John K. L. Ho. An interactive knowledge-based cad system for mould design in injection moulding processes. *The International Journal of Advanced Manufacturing Technology*, 17(1):27–38, 2001.

[47] 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.

[48] G.J. Muller. Design objectives and design understandability, 2007.

[49] M. M. Olthof. *Identifying and formalizing routine design problems in industrial settings*. PhD thesis, University of Twente, Faculty of Engineering Design, OPM-877, 2008.

[50] M. M. Olthof, J.M. Jauregui-Becker, H. Tragter, and F.J.A.M. van Houten. Knowledge acquisition of routine design problems in industrial settings. In *10th European Conference on Knowledge Management*, Vicenza, Italy., 2009.

[51] S. Orsborn, J. Cagan, R. Pawlicki, and R. Smith. Creating cross-over vehicles: defining and combining vehicle classes using shape grammars. *Artificial Intelligence in Engineering Design and Manufacturing*, 20(3):217–246, 2006.

[52] G. Pahl, W. Beitz, J. Feldhusen, and K. H. Grote. *Engineering Design: A Systematic Approach*. Springer, 2007.

[53] P. Y. Papalambros and D. J. Wilde. *Principles of Optimal Design- Modeling and Computation*. Cambridge University press, 2000.

[54] Van Parunak. Case grammar: A linguistic tool for engineering agent-based systems. *Unpublished white paper at http://www.iti.org/ van/ asegram.ps*, 1995.

[55] C. Petzold. *Programming Microsoft Windows with C#*. Microsoft Press, 2002.

[56] 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.

[57] W. O. Schotborgh. *Knowledge engineering for design automation*. University of Twente, 2009.

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

[59] W. O. Schotborgh, F. G. M. Kokkeler, H. Tragter, and Fjam 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.

[60] W. O. Schotborgh, H. Tragter, F. G. M. Kokkeler, Fjam van Houten, and T. Tomiyama. Towards a generic model of smart synthesis tools. *Proceedings of the CIRP Design Seminar 2007*, 2007.

[61] W.O. Schotborgh, M.H.L. Roring, D.R. Grigoras, H. Tragter, F.G.M. Kokkeler, and F.J.A.M. van Houten. A development methodology for parametric synthesis tools. In *ASME IDETC&CIE*, 2007.

[62] R. N.S. Schumacher. *Design formulas for plastic engineers*. Hanser Publishers, 2004.

[63] K. Shea and J. Cagan. The design of novel roof trusses with shape annealing: assessing the ability of a computational method in aiding structural designers with varying design intent. *Design Studies*, 20:3–23, 1999.

[64] K. Shea and J. Cagan. Languages and semantics of grammatical discrete structures. *AI EDAM*, 13(4):241–251, 1999.

[65] H. Simon. The structure of ill structured problems. *Artificial Intelligence*, 4(3):181–201, 1973.

[66] Jaroslaw Sobieszczanski, S. Emiley Mark, S. Agte Jeremy, and R. Sandusky Jr Robert. Advancement of bi-level integrated system synthesis (bliss). Technical report, NASA Langley Technical Report Server, 2000.

[67] Autodesk Software. http://www.moldflow.com/stp/, 2009.

[68] A. Soman, S. Padhye, and M.I. Campbell. Toward an automated approach to the design of sheet metal components. *AI EDAM*, 17(03):187–204, 2003.

[69] B. Spence, L. Tweedle, and H. Dawkes. Visualisation - enhancing qualitative design. In *CHI '95*, 1995.

[70] P. Sridharan and M. I. Campbell. A grammar for function structures. *Proceedings of the ASME Design Engineering Technical Conference*, 3:41–55, 2004.

[71] A. C. Startling and K. Shea. A parallel grammar for simulation-driven mechanical design synthesis. *International Design Engineering Technical Conferences 2005*, 2(A):427–426, 2005.

[72] N. P. Suh. Axiomatic design theory for systems. *Research in Engineering Design*, 10, pp 189-209, 1998.

[73] N. P. Suh. Complexity in engineering. *Annals of CIRP*, 54(2): 581-598, 2005.

[74] N. P. Suh. *Complexity: Theory And Applications*. Mit-Pappalardo Series in Mechanical Engineering. Oxford University Press, 2005.

[75] S. Szykman and R. D. Sriram. Design and implementation of the web-enabled nist design repository. *ACM Trans. Internet Technol.*, 6(1):85–116, 2006.

[76] S. N. Talukdar, S. Sachdev, and E. Camponogara. A collaboration strategy for autonomous, highly specialized agents. *Proceedings of the SPIE, Symposium on Intelligent Systems & Advanced Manufacturing*, 1997.

[77] T. Tomiyama. Dealing with complexity in design: A knowledge point of view. *Design Methods for Practice*, pages 137–146, 2006.

[78] T. Tomiyama. Intelligent computer-aided design systems: Past 20 years and future 20 years. *Artificial Intelligence for Engineering Design, Analysis, and Manufacturing*, 21:27–29, 2007.

[79] T. Tomiyama and H. Yoshikawa. Extended general design. *Centre For Mathematics and Computer Science, Technical Report CS-R8604*, 1986.

[80] K. Ueda. Synthesis and emergence - research overview. *Artificial Intelligence in Engineering*, 15:321–327, 2001.

[81] D. G. Ullman. *The Mechanical Design Process*. McGraw-Hill, 1992.

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

[83] K. Ulrich and S. Eppinger. *Product Design and Development*. McGraw-Hill, 2008.

[84] Y. Umeda and T. Tomiyama. Fbs modeling: Modeling scheme of function for conceptual design. In *Working Papers of the 9th Int. Workshop on Qualitative Reasoning About Physical Systems*, pages 271–278, Amsterdam, 1995.

[85] Y. Umeda and T. Tomiyama. Functional reasoning in design. *AI in Design*, 12(2):41–48, 1997.

[86] H. Yoshikawa. General design theory and a cad system. *Man-Machine Communication in CAD/CAM*, 1981.

[87] W. J. Zhang, Y. Lin, and N. Sinha. On the function-behavior-structure model for design, 2005.

[88] Y. Zhang, E. K. Antonsson, and A. Martinoli. Evolutionary engineering design synthesis of on-board traffic monitoring sensors. *Research in Engineering Design*, 19(2):113–125, 2008.

[89] L. Zhen, T. Liyong, W. Michael Yu, and W. Shengyin. Shape and topology optimization of compliant mechanisms using a parameterization level set method. *Journal of Computation in Physics*, 227(1):680–705, 2007.

Part V

Appendices

Appendix A

TARD and ToSE Example

A.1 Equation Generator

This example concerns the generation of equations. The idea is to have a TARD model capable of deriving expressions like the following:

$$a + b \cdot c - d = e/f$$

$$p_1 + p_2 \geq p_3 \cdot p_4 - p_5$$

One application of this model is the development of algebraic equations matching a predefined data set.

A.2 TARD Model

The appearance of an equation directly resembles its topology, which consists of variables that are related by operators. Therefore, the TARD model can be assembled by defining a single element class which represents all variables. Due to the fact that the operators relate two variables with each other, the choice is to denote a C-relation for each of the different operators. In a simple structure like this, the proximity relations deduce their contribution to the boundary elements from the element cardinality of the parent element, which is here equal to one.

$$e_L^{prox} = e_R^{prox} = e_{eq} = 1$$

Because the operators relate the variables in a one on one fashion, the cardinalities i and j are equal to one in each relation. All other cardinalities are unknown. This can be avoided by splitting the element denoting the variables into two elements, representing the parameters on the left and right hand side of

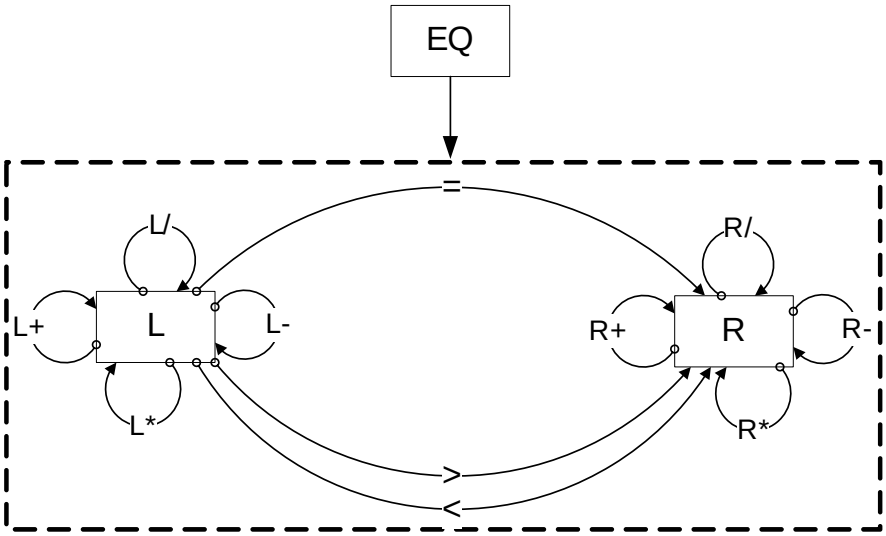


Figure A.1: *TARD model of the design of equations*

the equation respectively. The corresponding diagram is shown in Figure A.1. In this way, it is only possible to select between either one of =, < or > once.

A.3 ToSE Equations

Using the structure in Figure A.1, two balance and vertical equations can be derived. Since both elements are boundary elements, the proximity relations have to taken into account. The balance equations become:

$$L : j_{L+} \cdot l_{L+} + j_{L-} \cdot l_{L-} + j_{L*} \cdot l_{L*} + j_{L/} \cdot l_{L/} + e_L^{prox} =$$
$$i_{L+} \cdot l_{L+} + i_{L-} \cdot l_{L-} + i_{L*} \cdot l_{L*} + i_{L/} \cdot l_{L/} + i_{=} \cdot l_{=} + i_{>} \cdot l_{>} + i_{<} \cdot l_{<}$$

$$R : i_{R+} \cdot l_{R+} + i_{R-} \cdot l_{R-} + i_{R*} \cdot l_{R*} + i_{R/} \cdot l_{R/} + i_{=} \cdot l_{=} + i_{>} \cdot l_{>} + i_{<} \cdot l_{<} =$$
$$j_{R+} \cdot l_{R+} + j_{R-} \cdot l_{R-} + j_{R*} \cdot l_{R*} + j_{R/} \cdot l_{R/} + e_R^{prox}$$

With i = 1 and j = 1 this simplifies to:

$$L : 1 = l_{=} + l_{>} + l_{<}$$

$$R : l_{=} + l_{>} + l_{<} = 1$$

These balance equations resemble the limitations in using the operators $=$, $>$ and $<$. This proves the correct integration of the contains into the diagram by splitting the system into two elements. Accordingly, the vertical equations for L and R are:

$$e_L = [i_{L+} \cdot l_{L+} + i_{L-} \cdot l_{L-} + i_{L*} \cdot l_{L*} + i_{L/} \cdot l_{L/} + i_{=} \cdot l_{=} + i_{>} \cdot l_{>} + i_{<} \cdot l_{<}] \cdot e_{EQ}$$

$$e_R = [j_{R+} \cdot l_{R+} + j_{R-} \cdot l_{R-} + j_{R*} \cdot l_{R*} + j_{R/} \cdot l_{R/} + j_{=} \cdot l_{=} + j_{>} \cdot l_{>} + j_{<} \cdot l_{<}] \cdot e_{EQ}$$

With i , j and e_{EQ} equal to 1, this simplifies to:

$$e_L = l_{L+} + l_{L-} + l_{L*} + l_{L/} + l_{=} + l_{>} + l_{<}$$

$$e_R = l_{R+} + l_{R-} + l_{R*} + l_{R/} + l_{=} + l_{>} + l_{<}$$

The result is a set of 3 equations (the balance equations are the identical) and 13 unknowns. However, since all cardinalities $l_{=}$, $l_{>}$ or $l_{<}$ can be equal to 1, the balance equation can be used to calculate two cardinalities if one is known. Thus, this is an under-constraint with 8 degrees of freedom.

A.4 Generating Solutions

As shown to Figure A.3, a sequence can be generated with the Local Grammar Method. Accordingly, the values for all cardinalities l of each C-relation are calculated by using the sequence equations. Figure A.2 shows the available rules for the local grammar generation, where rule 1 to 7 apply for the element L and rule 8 to 11 for element R. Figure A.3 shows the stepwise generation of a possible solution.

The resulting sequence of the example gives results in:

$$l_{L+} = 1, \quad l_{L-} = 2, \quad l_{L*} = 1, \quad l_{L/} = 2, \quad l_{R+} = 0, \quad l_{R-} = 0$$

$$l_{R*} = 1, \quad l_{R/} = 0, \quad l_{=} = 1, \quad l_{<} = 1, \quad l_{>} = 1$$

By denoting the instances of both, L and R, as equation parameters p, the equation generated by this example yields:

$$p_1 \cdot p_2 + p_3 \cdot p_4 = p_5 \cdot p_6$$

In case $p1 = p2 = a$, $p3 = p4 = b$ and $p5 = p6 = c$, this equation becomes the equation of Pythagoras:

$$a^2 + b^2 = c^2$$

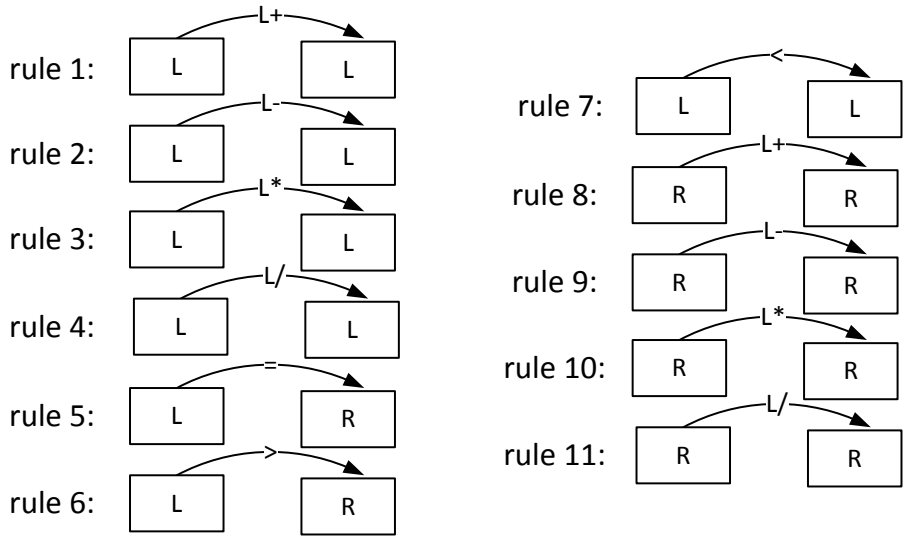


Figure A.2: *Rues in equation generator grammar*

Step	Current element	Grammar rules	Sequence
1	L	1 , 2 , 3 , 4 , 5 , 6 , 7	L*
2	L	1 , 2 , 3 , 4 , 5 , 6 , 7	L*-L+
3	L	1 , 2 , 3 , 4 , 5 , 6 , 7	L*-L+-L*
4	L	1 , 2 , 3 , 4 , 5 , 6 , 7	L*-L+-L*-=
5	R	8 , 9 , 10 , 11	L*-L+-L*-=-R*
6	R	finish	

Figure A.3: *Example sequence generation*

Appendix B

ToSE for CSIM

B.1 TARD Model

B.1.1 Elements

The base elements of this TARD model are the point elements: blue, green and brown. Combination of these points are used to create absorber channel, connector channel or input-/output channel. Channel elements are connected to among each others to form the abstraction-group cooling circuit. Finally, the entire cooling system is represented by the zero level element.

B.1.2 C-Relations

The first C-relation represents the connection of multiple cooling circuits into a single cooling system. The abstractiongroup described by channels is significantly more complex. The assembly of each circuit begins with an input channel. However, this channel can either be connected to a connector channel or directly to an absorber channel, represented by $C2$ and $C3$. For the output channel, the situation is the other way round, because it is connected from either of the two channels, as constituted by $C8$ and $C9$. The two channel elements might be either connected to itself respectively or with each other in both directions, ensuring a limitless variety of possible combinations.

B.1.3 Abstraction-groups

The abstraction-group describing the assembly of an input channel element contains two alternative c-relations connecting the brown point to a green or blue point. The c-relation in the abstraction-group of the absorber channel only connects to blue points, since all parts of this channel type fulfill the task of heat

absorption. Connector channel are assembled by connecting green points or green points with a blue point. Finally, the assembly of the output channel can start with a blue or green point, which is connected to a brown point by $C16$ or $C17$.

In summary, the network is composed of 6 complex abstractiongroups. These groups are distributed over 3 levels of abstraction and contain 14 element classes. All cardinalities i and j are equal to 1.

B.2 Balance equations

The proximity relations for the local cardinality of each boundary element is 1. This indicates that the sequence of each abstraction-group begins with a single, non-parallel element instance. Five balance equations can be established for the levels 1 and 2:

CoolingCircuit:

$$e^{prox} + j_{C1} \cdot l_{C1} = i_{C1} \cdot l_{C1} + e^{prox}$$

InputSegment:

$$e^{prox} = i_{C2} \cdot l_{C2} + i_{C3} \cdot l_{C3}$$

AbsorberChannel:

$$j_{C2} \cdot l_{C2} + j_{C4} \cdot l_{C4} + j_{C7} \cdot l_{C7} = i_{C4} \cdot l_{C4} + i_{C5} \cdot l_{C5} + i_{C9} \cdot l_{C9}$$

ConnectorChannel:

$$j_{C3} \cdot l_{C3} + j_{C5} \cdot l_{C5} + j_{C6} \cdot l_{C6} = i_{C6} \cdot l_{C6} + i_{C7} \cdot l_{C7} + i_{C8} \cdot l_{C8}$$

Outputsegement:

$$j_{C8} \cdot l_{C8} + j_{C9} \cdot l_{C9} = e^{prox}$$

With $e^{prox} = 1$ and $i = j = 1$, these equations can be writen as:

CoolingCircuit:

$$l_{C1} = l_{C1}$$

InputSegment:

$$1 = l_{C2} + l_{C3}$$

AbsorberChannel:

$$l_{C2} + l_{C4} + l_{C7} = l_{C4} + l_{C5} + l_{C9}$$

ConnectorChannel:

$$l_{C3} + l_{C5} + l_{C6} = l_{C6} + l_{C7} + l_{C8}$$

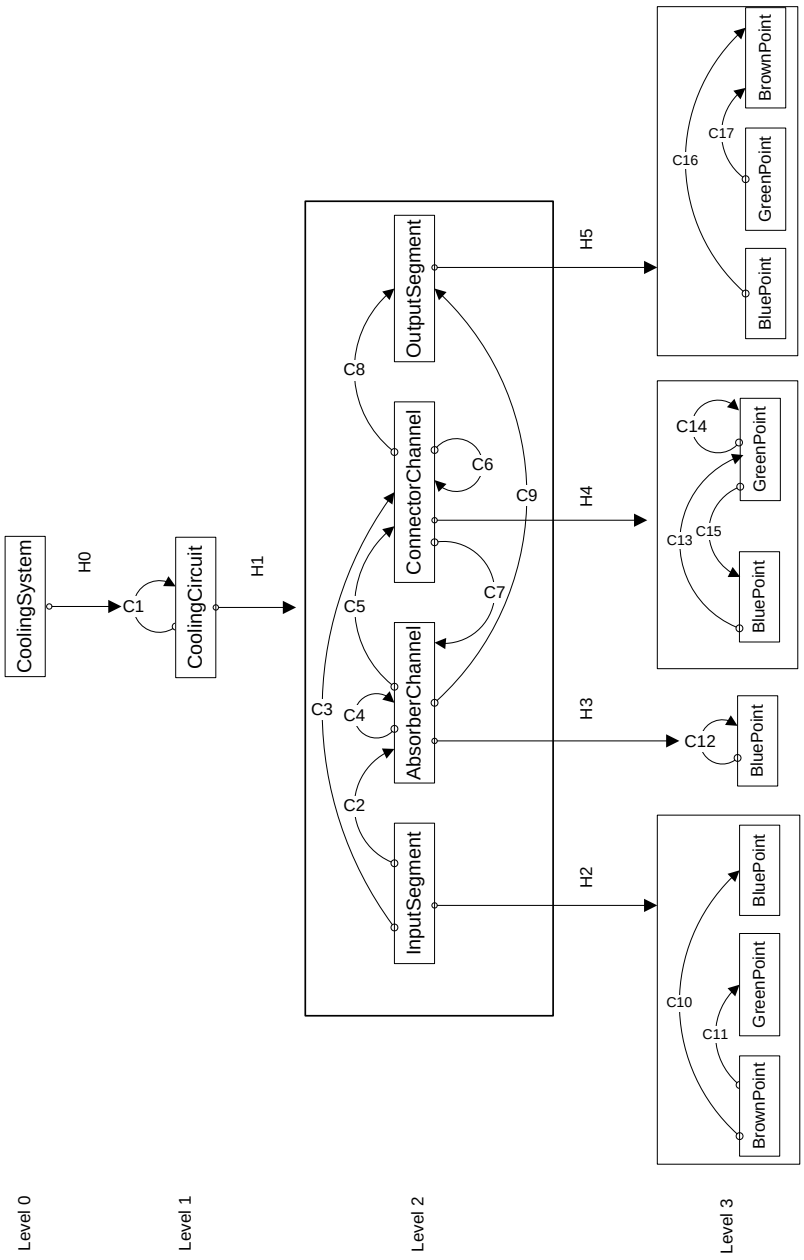


Figure B.1: *TARD model of CSIM problem.*

Outputsegement:

$$l_{C8} + l_{C9} = 1$$

The determination of the balance equation for the 3^{rd} level requires extra attention:

- Given that the three different point types are repeated in different abstraction-groups, each point type is treated as a different element class.
- As the abstraction-groups *H2* and *H5* connect blue or green points with brown points, each of these requires two proximity relations. For abstraction group *H2* this is:

  $$e^{prox}_{blue} = 1 - l_{C11}$$

  and

  $$e^{prox}_{green} = 1 - l_{C10}$$

  For abstraction-group *H5* this is:

  $$e^{prox}_{blue} = 1 - l_{C17}$$

  and

  $$e^{prox}_{green} = 1 - l_{C16}$$

- As *H4* is a complex abstraction-group, and the sequences can either begin or end with any of the elements, it is not possible to define proximity relations. This increases the uncertainty of the design problem by adding an unknown cardinality.

By taking this considerations, the balance equation of *H2* are:

BrownPoint:

$$e^{prox} = i_{C10} \cdot l_{C10} + i_{C11} \cdot l_{C11}$$

BluePoint:

$$j_{C10} \cdot l_{C10} = e_(bp)^{prox}$$

GreenPoint:

$$j_{C11} \cdot l_{C11} = e^{prox}_{gp}$$

With $e^{prox} = 1$, $e^{prox}_{gp} - l_{C10} = 1$, $e^{prox}_{bp} - l_{C11} = 1$, and $i = j = 1$, these equations become:

BrownPoint:

$$1 = l_{C10} + l_{C11}$$

BluePoint:

$$l_{C10} = 1 - l_{C11}$$

GreenPoint:

$$l_{C11} = 1 - l_{C10}$$

As it can be noticed, this equations are equivalent, which results in one equation representing the balance equation for all this elements.

For ***H3***, the balance equation is:

BluePoint:

$$e^{prox} + j_{C12} \cdot l_{C12} = i_{C12} \cdot i_{C12} + e^{prox} \Rightarrow 1 + l_{12} = l_{12} + 1$$

Taking into consideration the $3^r d$ aspect mentioned above, the balance equations of the abstraction-group ***H4*** are:

BluePoint:

$$e^{prox}_{bp} + j_{C15} \cdot l_{C15} = i_{C13} \cdot l_{C13} \Rightarrow e^{prox}_{bp} + l_{C15} = l_{C13}$$

GreenPoint:

$$e^{prox}_{gp} + j_{C13} \cdot l_{C13} + j_{C14} \cdot l_{C14} = i_{C14} \cdot l_{C14} + i_{C15} \cdot l_{C15} \Rightarrow e^{prox}_{gp} + l_{C13} + l_{C14} = l_{C14} + l_{C15}$$

The balance equations of ***H5*** are:

BluePoint:

$$e^{prox}_{bp} = i_{C16} \cdot l_{C16} \Rightarrow 1 - l_{17} = l_{16}$$

GreenPoint:

$$e^{prox}_{gp} = i_{C17} \cdot l_{C17} \Rightarrow 1 - l_{16} = l_{17}$$

BrownPoint:

$$i_{C16} \cdot l_{C16} + i_{C17} \cdot l_{C17} = e^{prox} \Rightarrow l_{16} + l_{17} = 1$$

Again here, all three equations are the same.

B.3 Vertical equations

The vertical are assembled for the base elements by considering that for all c-relations $i = j = 1$.

In abstraction-group ***H2***:

$$e_{brownPoint_{H2}} = (l_{C10} + l_{C12}) \cdot (l_{C2} + l_{C3}) \cdot l_{C1}$$
$$e_{greenPoint_{H2}} = l_{C11} \cdot (l_{C2} + l_{C3}) \cdot l_{C1}$$
$$e_{bluePoint_{H2}} = l_{C10} \cdot (l_{C2} + l_{C3}) \cdot l_{C1}$$

In abstraction-group *H3*:

$$e_{bluePoint_{H3}} = l_{C12} \cdot (l_{C4} + l_{C5} + l_{C9}) \cdot l_{C1}$$

In abstraction-group ***H4***:

$$e_{bluePoint_{H4}} = l_{C13} \cdot (l_{C8} + l_{C6} + l_{C7}) \cdot l_{C1}$$
$$e_{greenPoint_{H4}} = (l_{C14} + l_{C15}) \cdot (l_{C8} + l_{C6} + l_{C7}) \cdot l_{C1}$$

In abstraction-group ***H5***:

$$e_{brownPoint_{H5}} = (l_{C16} + l_{C17}) \cdot (l_{C8} + l_{C9}) \cdot l_{C1}$$
$$e_{greenPoint_{H5}} = l_{C17} \cdot (l_{C8} + l_{C9}) \cdot l_{C1}$$
$$e_{bluePoint_{H5}} = l_{C16} \cdot (l_{C8} + l_{C9}) \cdot l_{C1}$$

B.4 Summary

The ToSE consists of 18 equations, composed of 9 balances equations and 9 vertical equations. There are 28 cardinalities (17 l, 2 proximity relations and 9 element global cardinalities). As this design problem has 10 degrees of freedom and 2 complex abstract groups, it presents both time-dependent combinatorial complexity and time-independent imaginary complexity.

Appendix C

Generic CDS-CS Implementation

C.1 TARD Implementation

The architecture of the TARD implementation consist of two types of classes for each of the building block, namely, a *building block* class and a *building block-instance* class. For example, Figure C.1 shows that the building block *element* is implemented by an *element class* and an *element-instance class*. By doing so, elements in the problem class are instances of the generic elements class (1^{st} order instances), while elements in the problem solution are instances of the element-instance class (2^{nd} order instances). Furthermore, each of this classes has variables describing the building blocks cardinalities, references and parameters. This is shown in Figure C.2, where a simple UML diagram of the six topology classes is presented. As shown, an super class "ModelComponent" accommodates the building blocks classes. Figure C.3 shows the class description of these building blocks.

By using this architecture, a problem class is represented by making a 1^{st} order network, while problem instances and problem solutions are instances of the 2^{nd} order network. For the elements this means that the amount of 1^{st} order instances equals the amount of component types and abstraction-groups, whereas the amount of 2^{nd} order instances equals the sum of all element cardinalities within the 1^{st} order instances.

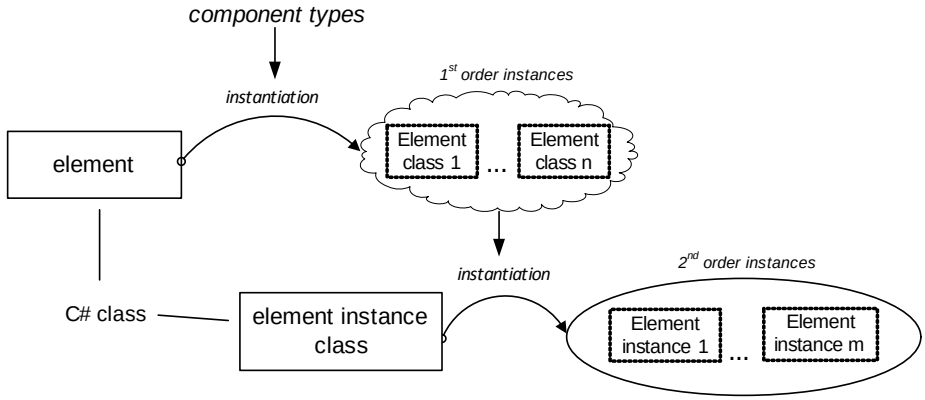


Figure C.1: *Architecture of the computer implementation of TARD building blocks: class and instance.*

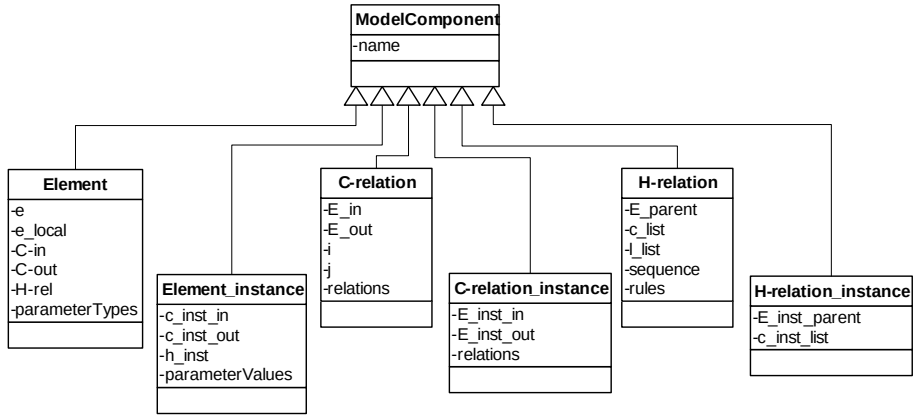


Figure C.2: *UML diagram of basic TARD building blocks.*

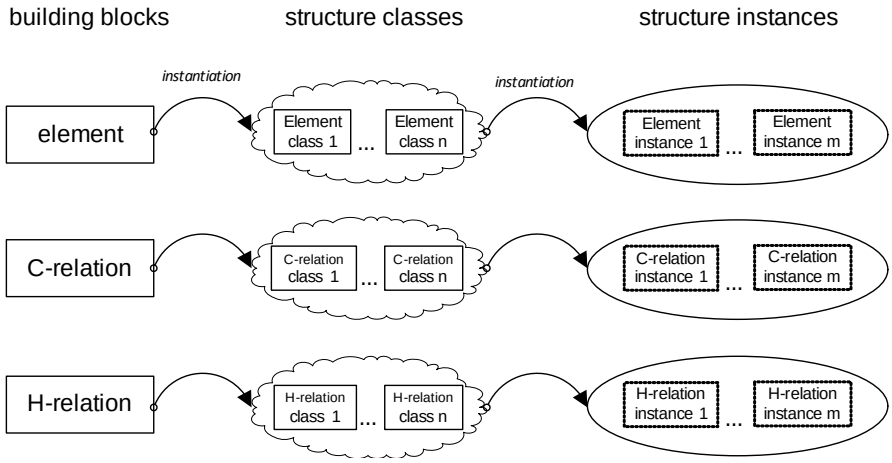


Figure C.3: *Class structure of TARD building blocks*

C.2 ToSE Implementation

ToSE is implemented at the hand of three classes, which are shown in Figure C.4. These classes are:

- ***LocalCardinalityFunction***: Objects of this class model the local cardinality of an element. A variable identifies if the cardinality is derived using incoming (IN) or outgoing (OUT) C-relations. It has a list with references that allows it to calculate the local cardinality of an element.
- ***BalanceEquation***: Objects of this class point to elements. It has two instances of the localCardinalityEquation, one based on incoming C-relations and another based on outgoing C-relations.
- ***VerticalEquation*** : Models the global cardinality of an element. It has a list of localCardinalityFunctions. The number of entries in that list depends on the level of abstraction of its element. It has a method to determine its value at the hand of this list.

In order to illustrate how this classes allow the implementation of ToSE, Figure C.5 shows the ToSE objects of element E of the TARD model shown in the same figure. The *localCardinalityFunction* relates the *balanceEquation* and *verticalEquation* to the cardinalities e,e_l, i, j, l, which are defined in their corresponding building blocks (element, C-relations and H-relations).

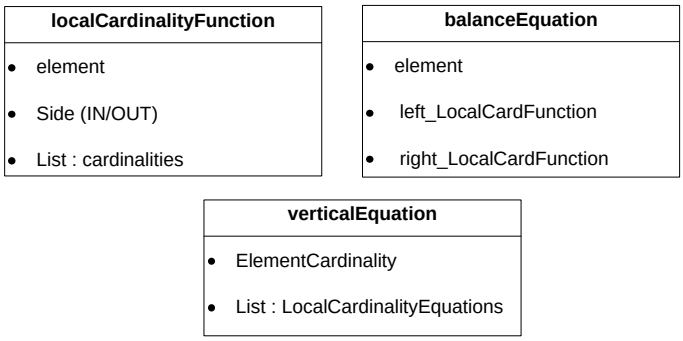


Figure C.4: *Diagrams of the equation and cardinality classes*

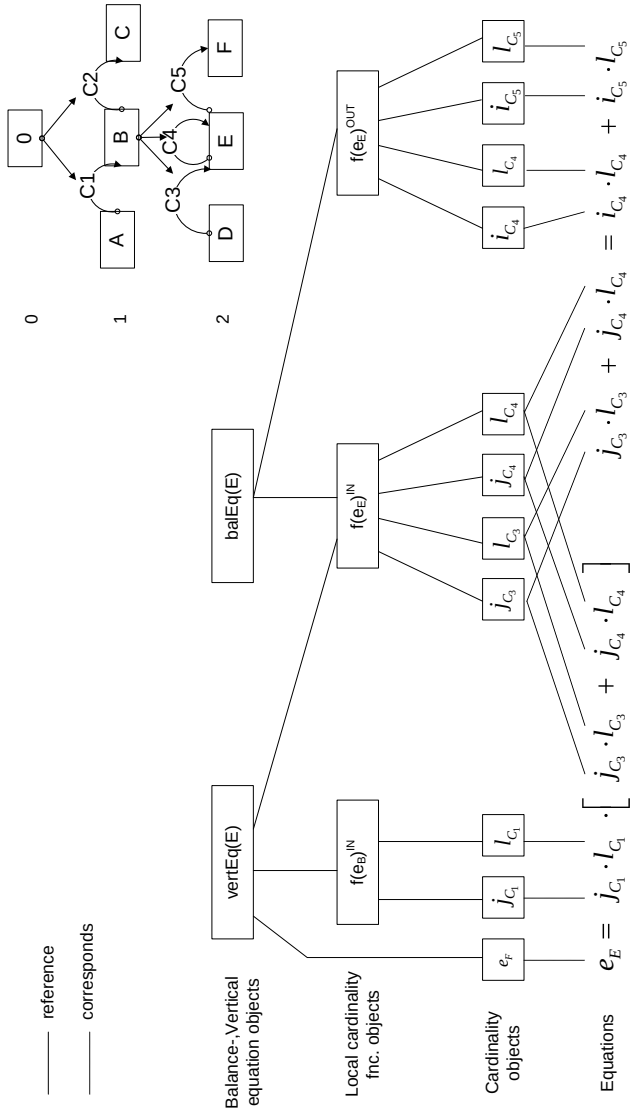


Figure C.5: *Object references (pointers) for ToSE equations.*