



HOMERO MAURÍCIO SCHNEIDER

SETTING UP A BACKTRACK-FREE CUSTOMISATION PROCESS FOR PRODUCT FAMILIES

*ESTABELECENDO UM PROCESSO DE CUSTOMIZAÇÃO LIVRE DE RETROCESSOS PARA FAMÍLIAS DE
PRODUTOS*

CAMPINAS
2014



UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Elétrica e Computação

HOMERO MAURÍCIO SCHNEIDER

SETTING UP A BACKTRACK-FREE CUSTOMISATION PROCESS FOR PRODUCT FAMILIES

ESTABELECENDO UM PROCESSO DE CUSTOMIZAÇÃO LIVRE DE RETROCESSOS PARA FAMÍLIAS DE PRODUTOS

Thesis presented to the School of Electrical and Computation Engineering of the State University of Campinas in partial fulfillment of the requirements for the Ph.D. degree in Electrical Engineering, in the area of Telecommunication and Telematics.

Tese apresentada à Faculdade de Engenharia Elétrica e Computação da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Engenharia Elétrica, na área de Telecomunicações e Telemática.

Supervisor/Orientador: YUZO IANO

Este exemplar corresponde à versão final da tese defendida pelo aluno Homero Maurício Schneider, e orientado pelo Prof. Dr. Yuzo Iano

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

Schneider, Homero Mauricio, 1953-
Sch58s Setting up a backtrack-free customisation process for product families /
Homero Mauricio Schneider. – Campinas, SP : [s.n.], 2014.

Orientador: Yuzo Iano.
Tese (doutorado) – Universidade Estadual de Campinas, Faculdade de
Engenharia Elétrica e de Computação.

1. Customização em massa. 2. Projeto auxiliado por computador. 3.
Inteligência artificial. 4. Representação do conhecimento. 5. Produtos novos -
Projetos. I. Iano, Yuzo, 1950-. II. Universidade Estadual de Campinas. Faculdade
de Engenharia Elétrica e de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Estabelecendo um processo de customização livre de retrocessos
para famílias de produtos

Palavras-chave em inglês:

Mass customization

Computer-aided design

Artificial Intelligence

Knowledge representation

New products - Design

Área de concentração: Telecomunicações e Telemática

Titulação: Doutor em Engenharia Elétrica

Banca examinadora:

Yuzo Iano [Orientador]

David Bianchini

Adão Boava

Rangel Arthur

Vicente Idalberto Becerra Sablón

Data de defesa: 25-04-2014

Programa de Pós-Graduação: Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE DOUTORADO

Candidato: Homero Maurício Schneider

Data da Defesa: 25 de abril de 2014

Título da Tese: "Setting Up a Backtrack-free Customisation Process for Product Families"

Prof. Dr. Yuzo Iano (Presidente): _____

Prof. Dr. David Bianchini: _____

Prof. Dr. Adão Boava: _____

Prof. Dr. Rangel Arthur: _____

Prof. Dr. Vicente Idalberto Becerra Sablón: _____

Abstract

Product family is a key concept in the area of mass customisation. Although the design of a product family is a difficult and challenging task, to derive members of the product family to meet the requirements of individual customers can be a routine design task. In this work, we propose a formal approach to model the customisation of product families that achieves this goal. In fact, this approach can be seen as a theory on the customization of product families. It is based on a knowledge framework for the representation of product families, which combines a generic product structure and a constraint network extended with design functions. The method for deriving members of the product family is a two-stage instantiation process. First, a solution to the constraint network model consistent with the customer requirements is found. Next, this solution is used to transform the generic product structure into a specific structure that corresponds to a member of the product family. In this work, we prove that if the constraint network model extended with design functions satisfies a few modelling conditions, then to find solutions become a backtrack-free process. Although there are other works in the literature that also claim to be backtrack-free, a remarkable fact about our approach is that we achieve this by the introduction of knowledge about the product family, instead of resorting to computational power and pre-processing as in those approaches. Another remarkable aspect of our approach is that components can be designed as part of the customisation process using the design functions. This implies that it is possible to have an efficient customisation process without compromising the flexibility of the product family. In the conclusion of this work, we argue that our approach can deal with customisation problems outside the product configuration area. Two appendices are also added to the thesis. One is a complete modelling of the Automatic Transfer Switch (ATS) product family using our approach. This example is used in the main body of the thesis to illustrate the concepts that are being introduced. The other appendix is the computational implementation of the first-stage of the customisation process of the ATS product family.

Resumo

Um conceito chave na área de customização em massa é o de família de produtos. Embora o projeto de uma família de produtos é uma tarefa difícil e desafiadora, derivar os membros da família de produtos para atender os requisitos de clientes individuais pode ser uma tarefa de design rotineira. Neste trabalho, propomos uma abordagem formal para modelar o processo de customização de famílias de produtos que atinge este objetivo. De fato, esta abordagem pode ser vista como uma teoria a respeito de customização de famílias de produtos. Ela é baseada em uma estrutura de conhecimento para a representação de famílias de produtos que combina uma estrutura genérica de produto e uma rede de restrições estendida com funções de design. O método para derivar os membros da família de produtos é um processo de instanciação com duas fases. Primeiramente, uma solução para o modelo de rede de restrição consistente com os requisitos do cliente é encontrada. Em seguida, esta solução é utilizada para transformar a estrutura genérica de produto em uma estrutura específica que corresponde a um membro da família de produtos. Neste trabalho, provamos que, se o modelo de rede de restrição estendida com funções de design satisfaz algumas condições de modelagem, então encontrar soluções se torna um processo livre de retrocessos. Embora existam outros trabalhos na literatura que também afirmam ser livre de retrocessos, um fato notável sobre a nossa abordagem é que conseguimos isso através da introdução de conhecimento sobre a família de produtos, em vez de recorrer ao poder computacional e pré-processamento como naquelas abordagens. Outro aspecto notável da nossa

abordagem é que os componentes podem ser projetados como parte do processo de customização através das funções de design. Isto implica que é possível dispor de um processo de customização eficiente sem comprometer a flexibilidade da família de produtos. Na conclusão deste trabalho, argumentamos que a nossa abordagem pode lidar com problemas de customização que estão fora da área de configuração de produtos. Dois apêndices também são adicionados à tese. Um deles é uma modelagem completa de uma família de produtos Chave de Transferência Automática (ATS) baseado em nossa abordagem. Este exemplo é utilizado no corpo principal da tese para ilustrar os conceitos que estão sendo introduzidos. O outro apêndice é uma implementação computacional do primeiro estágio do processo de customização da família de produtos ATS.

Contents

1	Introduction	1
1.1	Contributions of this work	3
1.2	General organization of this work	4
2	Conceptual background	5
2.1	Product families and product platforms	5
2.1.1	Market segmentation and product families	8
2.1.2	Modular architectures	9
2.2	Mass customisation	12
2.2.1	Problems with the elicitation process and the role of configurators	13
2.2.2	Flexibility in the mass customisation process	14
3	Product family modelling	17
3.1	Component types	17
3.2	The generic product structure	19
3.3	The constraint network extended with design functions	21
3.3.1	Variables	22
3.3.2	Constraints	25
3.3.3	Design functions	25
4	Product family customisation process	31
4.1	Finding solutions to the CN-F model	31
4.1.1	Instantiation patterns	35
4.1.2	Conditions for consistency	41
4.2	Transforming the GPS into physical models	44
4.3	Customisation of the ATS product family	45
5	Related approaches to the customisation of product families	51
6	Conclusions and Future Research	57
6.1	Applicability to the area of mass customisation	58
6.2	Implications for the routinization of the design activity	59
6.3	Further developments and future research	60
	References	63
	APPENDICES	69
I	Modelling the Automatic Transfer Switch	71

I.1	The ATS product family	71
I.2	The generic product structure	72
I.3	The constraint network model with extended functions	74
I.3.1	Variables	74
I.3.2	Constraints	80
I.3.3	Design functions	82
I.4	Instantiation patterns	89
I.5	The instantiation algorithm	91
II	Implementation of the ATS Customisation Process	93
Part I – The implementation		93
Part II – An example of the customisation process of the ATS product family		107

Acknowledgements

The development of this thesis was a long and essentially a lonely journey. However, there are those that along the way provided support. In particular, I would like to express my gratitude to Prof. Yuzo Iano for his readiness to support this journey, and for his patience and incentive along the way, and lately, for his concern so that all my effort could come to a successful end. I would like to extend my gratitude to the staff of the School of Electrical and Computation Engineering (FEEC) for their attention and help on many occasions. Furthermore, I am grateful to the Centre for Information Technology Renato Archer (CTI), to which I work for, for the time that I was allowed to devote myself to this thesis. I also would like to thank the financial agencies FINEP, CAPES, CNPq and FAPESP, in one way or another, support the development of this thesis. Finally, I would like to express my special gratitude to my family for their endurance along all the years I have been engaged on this work and for their expectations on the achievement of my goal.

List of Figures

Figure 1. Variety of portable stereos in the US market (from [18]).	6
Figure 2. Three models of Sony's Walkman.	7
Figure 3. Platforms for the Walkman (from [18]).	7
Figure 4. Market segmentation grid and the associated product platforms (based on [20]).	9
Figure 5. Architecture of trailers with different degrees of modularity (adapted from [21]).	10
Figure 6. Points of customers involvement in the mass customization process (modified from [32]).	15
Figure 7. The classification of component types in a product family.	19
Figure 8. Illustration of isomorphism and adherence.	20
Figure 9. The GPS (background) and the CN-F model (foreground) for the ATS product family.	22
Figure 10. An example of design function.	27
Figure 11. Master template for the wiring of the ATS product family.	28
Figure 12. The instantiation algorithm.	34
Figure 13. Instantiation patterns for the ATS example.	36
Figure 14. An instantiation patterns for the ATS example base on the d -function $fm'd$.	46
Figure 15. The architecture of the custom ATS.	49
Figure 16. The wiring component for the custom ATS.	50
Figure I-1. The ATS and its main components.	72
Figure I-2. The GPS (background) and the CN-F model (foreground) for the ATS product family	74
Figure I-3. Wire template for the ATS product family	79
Figure I-4. Space allowances between the switch and the sides of the enclosure	82
Figure I-5. The instantiation graphs for the ATS product family example.	89
Figure I-6. A simplified instantiation algorithm for CN-F models that satisfy Consistency Condition 1 and 2	91

Figure II-1. The <i>d</i> -function attached to the variable <i>Utility</i>	94
Figure II-2. The <i>d</i> -function attached to the variable <i>Engine-generator group</i>	95
Figure II-3. The <i>d</i> -function attached to the variable <i>Emergency load</i>	96
Figure II-4. The <i>d</i> -function attached to the variable <i>Neutral grounding</i>	97
Figure II-5. The <i>d</i> -function attached to the variable <i>Terminal block inclusion</i>	97
Figure II-6. The <i>d</i> -function attached to the variable <i>Switch specification</i>	98
Figure II-7. The <i>d</i> -function attached to the variable <i>Switch poles</i>	99
Figure II-8. The <i>d</i> -function attached to the variable <i>Control board configuration</i>	100
Figure II-9. The <i>d</i> -function attached to the variable <i>Terminal block amperage</i>	101
Figure II-10. The <i>d</i> -function attached to the composite variable <i>Enclosure</i>	102
Figure II-11. The <i>d</i> -function attached to the variable <i>Buzzer inclusion</i>	103
Figure II-12. The <i>d</i> -function attached to the variable <i>Enclosure hole</i>	103
Figure II-13. One view of the <i>d</i> -function attached to the variable <i>Wiring configuration</i>	104
Figure II-14. Second view of the <i>d</i> -function attached to the variable <i>Wiring configuration</i>	105
Figure II-15. Implementation of the control structure of the instantiation algorithm.....	106
Figure II-16. The Front Panel of LabVIEW showing all the variables used in the ATS configurator.	107
Figure II-17. Inputs regarding the <i>Utility</i>	108
Figure II-18. Input regarding the <i>Neutral grounding</i>	109
Figure II-19. Inputs regarding the <i>Emergency load</i>	109
Figure II-20. Inputs regarding the <i>Motor-generator group</i>	110
Figure II-21. Input regarding the <i>Terminal block inclusion</i>	110
Figure II-22. The solution for the customer requirements.	111

List of Tables

Table 1. Typology of modularity (from [3])	11
Table 2. Available transfer switches	24
Table 3. Available enclosures	45
Table 4. The input variables and their respective domains.....	47
Table I-1. Specifications of the available ATS enclosures.	76
Table I-2. Specifications of the available terminal blocks.....	77
Table I-3. Specifications of the available switches.....	78
Table I-4. Specifications of the cross-section area of the available wires.	79
Table I-5. Summary of the relations between constraints, <i>d</i> -functions and instantiation patterns	90

Chapter 1

Introduction

Product variety is a competitive strategy of companies to improve their market share by offering products tailored to specific market niches. However, an unplanned proliferation of products will cause the rising of the manufacturing costs of the company and create inefficiencies [1, 2]. To counter these negative effects, the concepts of product families and product platforms have proven to be a successful approach [3]. Among the major advantages that can be attributed to them is the reuse of components between the members of the product family and the reduction time for the development of new family members.

Product families have also been advocated to be a key concept in the mass customisation area [4, 5], but with some additional challenges. As we shall see in more details later, in contrast to mass production strategy, which provides a heterogeneous market with standard products, mass customization is a market strategy whose goal is to provide products that fit the needs of the customers individually, at costs comparable to mass production [6]. In this application, the mass customization company will have to design a product family that covers a delimited market segment, and to set up a flexible and stable mass customisation system that is capable of deriving members of the product family to meet the needs of individual customers in a responsive way [7]. The customers interact with the mass customisation system through a configurator, which consists of a software system that assists the customers in the process of specifying a product that meets their requirements [8, 9].

The capability of solving a configuration problem without backtracking is essential for the success of the mass customization company since it allows the customer to specify the product he

needs without going into dead ends. This happens whenever the requirements of the customer cannot lead to a consistent product. In this case, the customer will have to review previous decisions until a product specification that meets the requirements can be found. This may become a tedious interaction with the customer eventually given up. Despite its importance, in general, approaches to product configuration either cannot guarantee a backtrack-free process or will achieve this by some type of pre-processing, but running the risk of computational time or space explosion [10]. In this work, we propose an approach that takes advantage of the design process of product families to provide enough structure to the customization of the product family that it can become a backtrack-free process.

Certainly, the design of a product family is a “difficult and challenging task” [11], for it requires the development of multiple products at the same time keeping as much components in common to improve economy of scale, but without sacrificing variety or performance [12, 13]. However, it should not come as a surprise that after the product family is designed, deriving its members can be turned into a routine design task, one for which the problem solving requires little or no backtracking at all [14]. This claim follows from the fact that during the design process, designers acquired a great amount of knowledge about the structure and variability of the product family, and how the variable aspects of the product family are interrelated.

In the area of artificial intelligence, the task of deriving products that meet different sets of requirements is referred to as product configuration. It is well known that to improve the efficiency of the product configuration process it is necessary to use knowledge about the problem domain to constrain the design space and to guide the search process [15]. In this thesis, we propose a new approach based on a knowledge framework composed of two general models. A generic product structure (GPS) to represent the product family architecture, and a constraint network model extended with design functions (CN-F) that complement the GPS in delimiting the design space of the product family. The CN-F model is an extension of the classical constraint network (CN) model by associating design functions to the variables to generate their values during the customisation process. Members of the product family are derived from the knowledge framework as instantiations into two stages. First, an instantiation algorithm applies the design functions to find a solution for the CN-F model out of the customer requirements. Then, this solution is used to transform the GPS directly into a specific physical model that corresponds to a product family member.

Besides generating values to the variables of the CN-F model, the design functions play a crucial role in the problem-solving strategy for they establish dependency patterns between the variables, which guide the instantiation process. These structures reduce considerably the search for

solutions in the design space and eventual backtracking is confined to a subset of the variables used to express the customer requirements. However, the most important contribution of this work in this regard is the setting up of a few modelling conditions such that if the CN-F model satisfies them, the instantiation process becomes backtrack-free.

Turning the customisation of the product family into a backtrack-free process has a clear advantage in terms of problem-solving efficiency and in the simplicity of the control algorithm that is required. Nevertheless, in our approach, this gain in efficiency and simplicity does not compromise the flexibility of the product family. Because the design functions can be used to design components (in part or entirely) during the customisation process, the custom products can be tailored to the requirements of the customers. It also implies that the customisation process does not have to rely on pre-defined components. As we will argue in the conclusion of this work, because of this capability, our approach can be distinguished from product configuration approaches, as well as the other approaches reviewed within this work.

To illustrate the concepts that will be introduced along this work, we will consider the example of the Automatic Transfer Switch (ATS) product family. An ATS is an electronic device that senses the loss of power on the utility and promptly activates an emergency generator set to restore power to a vital load, such as emergency lights, security equipment, etc. The load is automatically transferred back to utility after its power has been restored. More details about the ATS product family are given in Appendix I.

1.1 Contributions of this work

In this work, we provide many contributions to the area of product configuration, or to product customization, in a broader sense. The following list summarizes the main achievements of our approach:

- It sets up a formal theory of the customization process of product families;
- Introduces a knowledge framework for the representation of for product families, which combines the GPS and the CN-F models, and a two stage process for deriving product family members;
- Introduces a formal definition for product families based on the GPS;
- Introduces the CN-F model as an extension to the classic constraint network (CN) model by design functions;
- Introduces the concept of instantiation patterns, based on the dependencies between the variables established by the design functions;

- It sets up a few conditions that if satisfied, deriving product family members becomes a backtrack-free process.
- The problem of product configuration is treated as a data flow process.

Although our approach has been developed in view of the mass customisation area, many other application areas, such as, software product lines and self-configuring products, can benefit from it as well.

1.2 General organization of this work

The remaining of this thesis is organized as follows. In Chapter 2, we set up the conceptual background of the approach, examining the concepts of product family, product platform and mass customisation. Along this chapter, we put our approach in perspective by clarifying its application in mass customization and delimiting its scope. In Chapter 3, we develop the framework for representing product families. We begin with the definition of component types, followed by a definition for the GPS and a definition of product families based on the GPS. Then, we deal with each element composing the CN-F model. In Chapter 4, we introduce the two-stage method for deriving product family members. It begins with the definition of our instantiation algorithm to find solutions for the CN-F model and its consistency issues. Next, we introduce the concept of instantiation patterns for the CN-F model. The chapter culminates with the definition of two consistency conditions for the CN-F model under which the instantiation process is backtrack-free. After that, we introduce the method for transforming the GPS into the physical model of a member of the product family. Finally, we illustrate the use of instantiation method on the ATS product family. In Chapter 5, we review the literature on approaches related to our proposal and discuss some aspects of our approach in comparison to them. Finally, in Chapter 6, we sum up the main advantages of our approach, make some comments about its applicability and give directions for further research.

Two appendices have been included as part of the thesis. In Appendix I, we present the complete and detailed modelling of the ATS product family. We also discuss the consistency conditions in connection to CN-F model of the ATS product family, and present a simplified version of the instantiation algorithm. In Appendix II, we present the implementation of our prototype configurator for the ATS product family. As a practical demonstration of the program, we run a example of customer requirements.

Chapter 2

Conceptual background

In what follows, we examine in more detail most of the concepts referred to in the introduction. However, this chapter is not intended to be a review on those concepts, but only to set up the conceptual background of our approach, to make clear the application area and the type of problem it is aimed at. First, we consider the concept of product families and product platform through an example. We also look at the relation of product family and market segmentation, and the role of modular architectures in the generation of product variety. Then, we examine the definition of mass customisation, the application area for which our approach has been developed. Regarding mass customisation, we focus on the elicitation process and the role of configurators in this process. Then we examine the major factors for the success of a mass customisation company, with emphasis on flexibility.

2.1 Product families and product platforms

As mentioned in the introduction, our approach is based on a knowledge framework for representing product families. In fact, we are assuming the product family as given and that the necessary knowledge about it is available from its design process. Hence, in this work, we will not be specifically concerned with the process and tools used for the development of product families, but only with its concept and main characteristics. The interested reader will find an extensive review of those subjects in [16, 17].

To clarify the concept of product families, it will be instructive to begin with an example. The Walkman is a Sony brand trade name for portable audio cassette players, which has achieved great

success in the 80s and early 90s. This example is interesting because it illustrates very well the proliferation of products that is possible with the concept of product families. Moreover, it also illustrates the concept product platform, which is closely related to the concept of product families.

In 1979, Sony launched its first model of the Walkman in Japan. Two years later, Sony launched the WM-2, the second generation of the device, in a worldwide advertising campaign. During the 80s, Sony introduced about 20 new models in the US market each year. For the models introduced from 1980 to 1988, the market life was 1.97 years on average. Figure 1 illustrates the variety of models in the US market from 1981 to 1991. The variety of new models introduced in that period was a response to customer needs, but was also driven by technological improvements. Along that decade, Sony dominated the personal portable players market in the world.

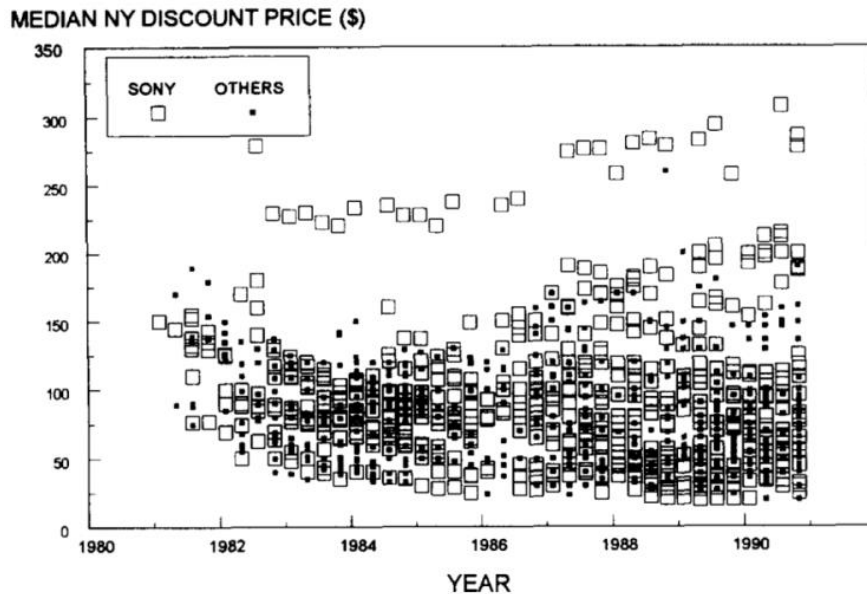


Figure 1. Variety of portable stereos in the US market (from [18])

In Figure 2, it is shown three of the models of the Walkman with their variations. The WM-2, with dimensions slightly larger than a cassette tape, was the second generation of Walkman launched by Sony and the most sold in the world. Another precursor of several models, the WM-20 was designed with a flat low-power engine that operated with only one 1.5 V AA battery, and its thickness was half of the WM-2. The first sporty model was the WM-F5. It made use of the same DD direct drive mechanism as the WMDD model; however, one of its distinguishing characteristics was to be waterproof.

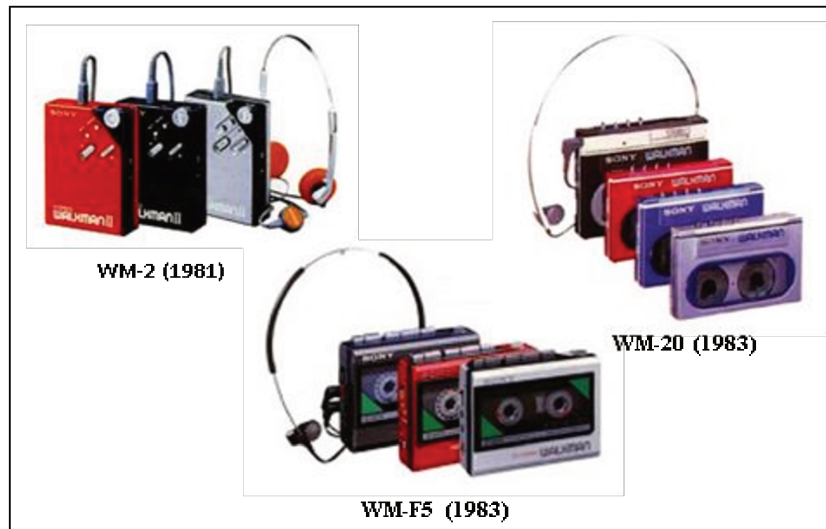


Figure 2. Three models of Sony's Walkman

Despite the apparent large diversity of models, most of the Walkman models were obtained from three basic platforms (models WM-2, WMDD and WM-20, shown in Figure 3) by small changes in the features, packaging and appearance of these platform models. However, those platforms were the outcome of major engineering efforts, giving rise to sub-families of Walkman models. As indicated in the figure, significant improvements in component technology were also made during the 80s. Basic components were standardized and a flexible manufacturing system was set up to produce all the models within a sub-family [18].

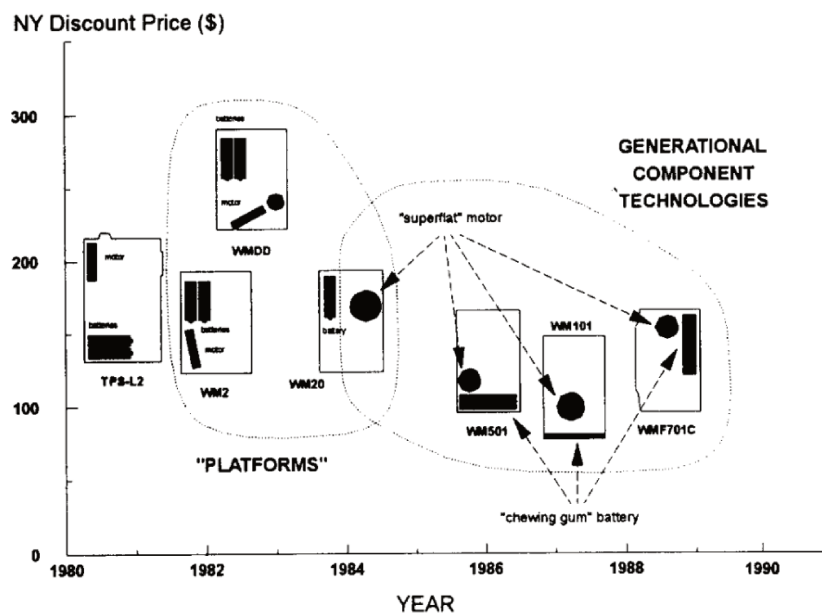


Figure 3. Platforms for the Walkman (from [18])

Based on the Walkman case discussed above, one obvious observation is that a product family is a set of related products. However, central to this concept is the fact that all members of product family are derived from a collection of assets, or product platform. According to Robertson and Ulrich [19], this shared collection of assets includes the components that are used to assemble the products, the embedded software, product family architecture, the equipment and process used to produce the components, the design teams and their know-how, etc.

Another definition for product platform is given in [20]. They define it as “a set of common components, modules or part from which a stream of products can be efficiently created and launched.” It should be noted that, based on this definition, if the core set is composed of just a few components, then it may be possible to create from the product platform a plethora of different functional products with little more in common than those components, all of them regarded (by definition) as part of the same family. For example, in the Black & Decker case described by Mayer and Lehnerd [20], a universal motor was the key component of the product platform. From this platform, a portfolio of tools as diverse as drills, sanders, grinders, saws etc. were developed. In our view, this poses some difficulties to apply the concept of product family to a mass customisation strategy. In our approach, one key asset in the product platform is the GPS. As we shall see in Section 3.2, this concept plays a central role in restricting the functional scope of the product family without compromising its capability to meet different customer requirements within a heterogeneous market segment.

2.1.1 Market segmentation and product families

The strategy behind a product platform is to “maximize market leverage from common technology.” This is accomplished by serving different but related market segments with family members derived from the product platform. To set up a robust product platform in association to a market strategy, Meyer and Lehnerd [20] proposed the platform-market grid tool shown in Figure 4. The major customer groups served by the products of a company are arranged horizontally into distinct market segments. These market segments are further subdivided into different tiers of price or performance, for example, low cost/performance, mid-range and high cost/performance. Each of the subdivisions of the resulting market grid is a market niche. Based on this grid, a company may establish a strategy of which niches it will serve at given time and with what product platforms, and how the market coverage of the product platform will evolve over time.

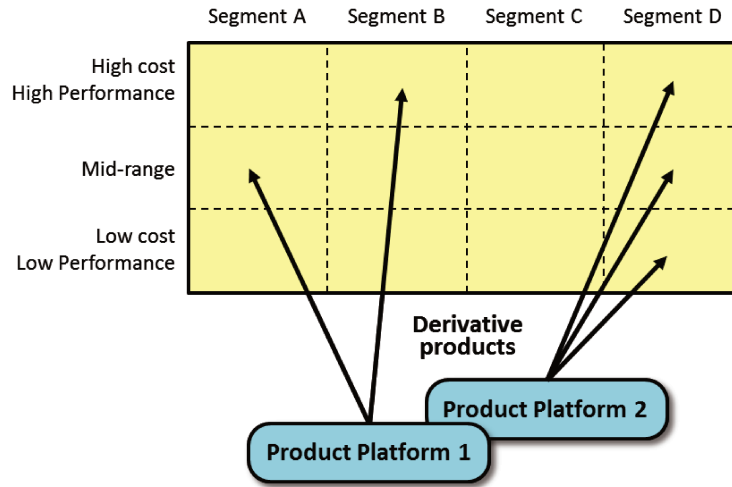


Figure 4. Market segmentation grid and the associated product platforms (based on [20])

However, within a single market niche the same product variant is being offered to a large number of customers, not all of them with the same needs. Thus, from a market point of view, a standard product is still being offered for every market niche, except that now the scope of each market segment has become more focused to the needs of a particular group of customers. However, the segmentation process could precede each market niche being subdivided further, hence reducing even the heterogeneity of the market segment. However, soon the number of product variants would become unmanageable and this market strategy enviable from the economical point of view. However, this exercise is interesting because this point stresses one important difference between using the product platform as a strategy for providing the market with a variety of products, and a mass customization strategy based on a product platform, which, in principle, derives products that meet the needs of individual customers. In this regard, a mass customisation market strategy may be considered as the limiting case of the market segmentation process, a condition where every customer is a different market niche. However, this possibility depends on the flexibility of the product family and the capabilities of the mass customisation system. We will return to this point in Subsection 2.2.2.

2.1.2 Modular architectures

A product is a collection of components assembled to realize the desired functions. Besides the physical arrangement of components, there is a correspondent conceptual arrangement of functional elements that defines how the product overall functions are brought about. A modular architecture is defined by Ulrich [21] as a condition for which the functional elements have a one-to-one mapping to physical components and a well-defined specification of the interfaces among interacting components.

However, products can have different degrees of modularity, ranging from a completely integrate to completely modular architecture. Thus, the definition given above should be understood as a condition for a highly modular architecture. In Figure 4, it is shown two examples of architectures with different degrees of modularity: a) a modular architecture with a one-to-one mapping and b) an integrated architecture where some functions are realized by more than one component, and some components are involved with more than one function.

Because in a modular architecture the couplings between components or modules (i.e., groups of components with identifiable functions) are greatly reduced, changes introduced to one component or module can be prevented from propagating to neighbouring ones. Thus, a modular architecture offers a great opportunity to promote variety. Actually, it is widely recognized in the literature that this is a fundamental concept for achieving product variety [22].

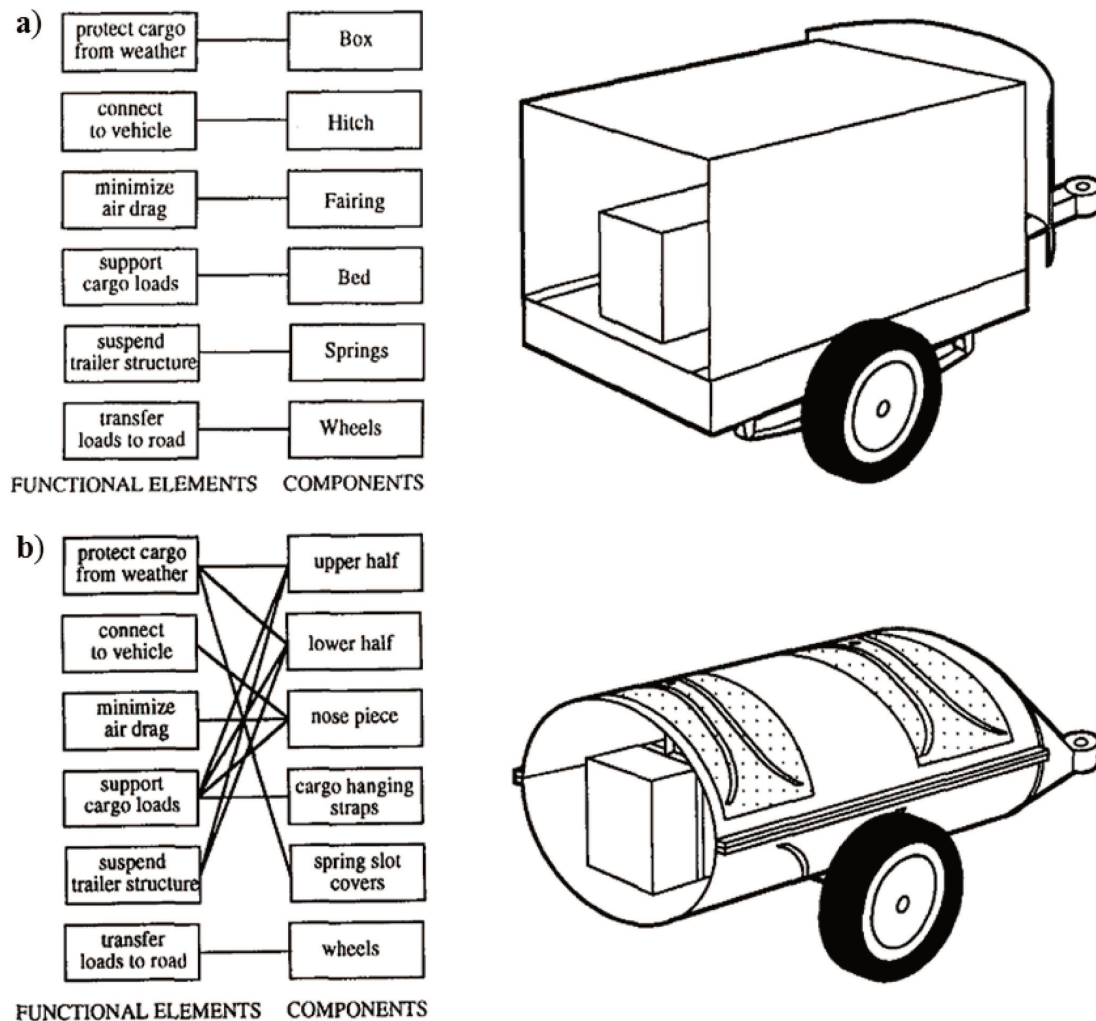
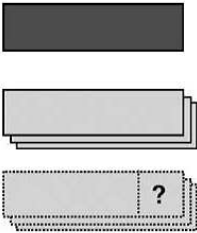
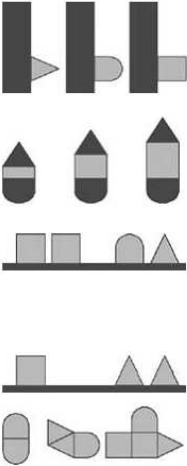
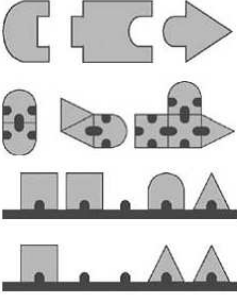


Figure 5. Architecture of trailers with different degrees of modularity (adapted from [21])

Table 1. Typology of modularity (from [3])

References	Classification criterion		Types of module/modularity
Pahl and Beitz (1984)	Stability of the function allocated to the component		<p>Basic and auxiliary modules implement functions that are common throughout the product family</p> <p>Special modules implement complementary and task-specific functions that do not need to appear in all the product variants</p> <p>Adaptive modules implement functions related to the adaptation to other systems and to marginal conditions</p>
Ulrich and Tung (1991)	How the final product configuration is built		<p>Component swapping modularity: product variants are obtained by swapping one or more components (▶, ◻, ◻) on the common product body (◻)</p> <p>Fabricate-to-fit modularity: product variants are obtained by changing a continuously variable feature (◻, ◻, ◻) within a given component</p> <p>Bus modularity: product variants are obtained by matching any selection of components from a set of component types (▶, ◻, ◻) with a component that has two or more interfaces (◻)</p> <p>Sectional modularity: product variants are obtained by mixing and matching in an arbitrary way a set of components (▶, ◻, ◻), as long as they are connected at their interfaces</p>
Ulrich (1995)	Nature of the interface between components		<p>Slot modularity: interfaces between different components are different (◻, ◻)</p> <p>Sectional modularity: all the components are connected via identical interfaces (◻)</p> <p>Bus modularity: special case of sectional modularity where there is a single component, the bus (◻), performing the connection function</p>

Different types of modularity have been identified, and a typology for modular structures has been brought forward. For example, Pahl and Beitz [23] identified modules as being basic, auxiliary, special or adaptive. Ulrich and Tung [24] identify types of modularity through which variety can be obtained, namely, component swapping, fabricate to fit, bus and sectional modularity. Based on the types of interfaces between components and their organization, Ulrich [21] also introduced a typology subdividing modular architectures into slot, bus and sectional. In Table 1, it is shown a summary of this typology. Although this typology is helpful for the understanding the nature of modular architectures and how product variety is entailed, it falls short of providing a modelling language to represent product families. This issue is the subject of our approach and will be presented in Chapter 3.

The classification of components introduced by Pahl and Beitz [23] brings to light one important point. A product family has a set of components that are common to all its members, and that define its main functions. We already discussed this point in relation to the definition of product platforms, calling attention to the implications of a core set with too few components. On the other hand, the variability within the product family is brought about by a set of special components. This distinction between components can be seen throughout the literature and are more frequently referred to as common and variant components. Since a product family with too many common components has low variability, and one that has too many variant components has higher manufacturing costs, the commonality/diversity issue and its optimization has been the focus of some works, for example, in [25].

2.2 Mass customisation

Differently from the prevailing mass production strategy, which supplies the market with standard products produced in high volumes and at low cost, mass customization is a market strategy which aims at providing the customers individually with products that fit their needs at a price comparable to standard mass-produced products [6]. In a more recent definition, Piller [7] stresses the main factors for the success of the approach. He defines mass customization as a “customer co-design process” for the manufacturing of products, for which “all operations are performed within a fixed solution space, characterized by stable but still flexible and responsive processes.” By co-design, he means the integration of the customers into a value creation process “by defining, configuring, matching, or modifying an individual solution.” By a solution space, we can understand the set of all products that can be derived from the product platform that is supporting the mass customisation system. A fixed and well-defined solution space is crucial for the stability and responsive of the mass customisation system. However, as will be discussed below, this tends to work against the flexibility.

Zipkin [26] identifies three key elements for a mass customisation system. An elicitation system that is capable of eliciting the customer requirements interactively. A flexible manufacturing system that is capable of producing the products the customers need and that can reduce “the trade-off between variety and productivity”. A logistic system that is capable of delivering the right product to the right customer. In this work, we will be concerned only with the elicitation process. This process includes the interactions with the customers to define their requirements, up to the point when the product that meets their requirements is completely specified. After that, the product specification may go through additional stages to transform it into a suitable representation for the manufacturing process.

An extensive review of the literature on mass customisation can be found in [27, 28]. In the next subsection, we will examine some of the difficulties associated to the elicitation process and the use of configurators as a tool for its automation. After that, we will characterise flexibility in the mass customization system in more details.

2.2.1 Problems with the elicitation process and the role of configurators

One of the main problems with the elicitation process is that, in general, customers do not know exactly what they need and, in this case, it may not be an easy task to help them to articulate their requirements. Moreover, they can easily become overwhelmed and get frustrated if too many options are provided to them during the elicitation process, a condition that has been called by Pine as “mass confusion” [29]. Even in the assumption that the customers are willing to go through the elicitation process, if they are exposed to technical choices, unless they have the necessary background, they may feel unsure whether the choices made will lead to products that meet their needs. In that situation, it is necessary to bridge the gap between the needs of the customers to the technical decisions necessary to specify a product. Because the elicitation process stands as the front end through which the customers interact with the mass customisation system, this is an area that deserves much attention [30, 31].

A configurator is a software tool that assists the customers during the elicitation process to specify products that meet their needs. Hence, it is a key technology for the success of the mass customization system is the configurator. A configurator comprises three main subsystems [9]:

1. A core configuration subsystem with an interface through which the customer can interact to define the product that meets his requirements.
2. A feedback subsystem that provides to the customer the relevant information for his decision-making. The visualization of the custom product and its price are typical forms of feedback.
3. A back-end subsystem that translate the specifications of the custom product into a production order with all the necessary documentation to start its manufacturing.

Two important problems that manifest themselves in the interaction of the customers with the mass customization system have their roots on the core configuration system. If every time the customer makes a decision, the problem solving mechanism of the core configuration system has to go through an extensive search process in the design space, it may become unresponsive for long periods of time. Moreover, if the problem-solving process is not backtrack-free, it may frequently reach dead ends, requiring the customer to review previous decisions. These characteristics will inevitably scare

the customers. Therefore, the availability of backtrack-free problem solving methods, which exhibits real time responses, is of great value for the development of configurators.

Besides its role in the interface, a configurator can also provide many important benefits to the mass customisation company. Reduction of the order fulfilment cycle time by the automatic generation of quotations, and accurate order entry are among those benefits [8, 3]. Because of their impact on the success of the mass customisation system, this is also an area of research of great interest.

The approach proposed in this work will provide a theoretical foundation over which backtrack-free configurators can be developed. Since we will be mainly focused on problem solving aspects of the customisation of product families, the results presented in this work will be more closely related to the core configuration subsystem mentioned above. Nevertheless, some aspects of the back-end subsystem will be supported to some extent. In Appendix II, we will present the implementation of a prototype configurator for the ATS product family, as a practical demonstration of our approach.

2.2.2 Flexibility in the mass customisation process

If the product is to meet the individual customer requirements, the manufacturing system must allow the introduction of the customer requirements at some point of the production process, called the point of customer involvement [32]. Points of entry can happen at different stages of the manufacturing process, each one enabling a different degree of customisation of the output product. In Figure 6, it is depicted the four main entry points along the manufacturing process. The earliest entry point is at the design stage. This means that the whole design process is carried out in view of the customer requirements. The output of the manufacturing process is a unique product that can fit exactly the customer needs. The second entry point corresponds to a condition where the design of an existing product is modified or adapted to meet the customer requirements. In this case, some of the product components will be modified or new ones may be designed to meet the customer requirements. The output of the manufacturing process is a product with some unique features or components. In the third entry point, the product is assembled from a finite collection of existing components. This corresponds to a typical product configuration process. In the fourth and last entry point, only minor adjustments are made, either by the setting of the product or the addition of accessory components. It should be noted that these entry points are not exclusive, for the customisation process can involve combinations of these entry points. For example, it could involve the redesign and fabrication of some components and the configuration of available components.

The first entry point corresponds to the Engineer-to-Order (ETO) strategy in the literature. In [33] it is argued that, if a company operates according to an ETO strategy, it does not imply that it conforms to the definition of mass customisation. Possibly, it will fail regarding the cost and responsiveness criteria. On the other hand, at the last entry point, the customisations are rather trivial and consist mostly of product bundling. A task left to the sales staff to do.

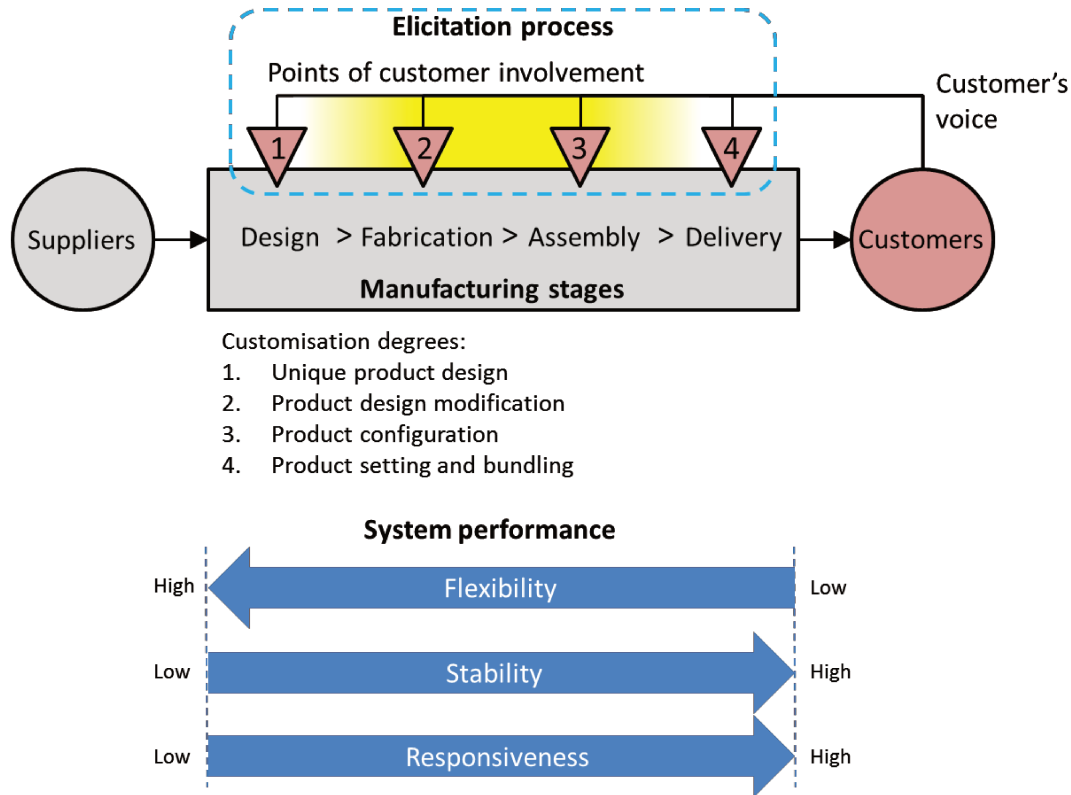


Figure 6. Points of customer involvement in the mass customization process (modified from [32])

Therefore, mass customisation approaches are most appropriately related and likely to occur in the middle entry points, as indicated in Figure 6 by the highlight. In this figure, it is also shown how the degree of flexibility, stability and responsiveness of the mass customization system are related to the entry points. By flexibility, it is meant how closely the mass customization system can meet the customer requirements within a heterogeneous market segment. By stability, it is meant the capability of the mass customization system to cope in a cost-effective way with the variety of products that must come out from it without having to go through adjustments. By responsiveness, it is meant the time it takes for the products to be delivered to the customers from the moment they are involved in the elicitation process. As indicated in the figure, while the flexibility will increase as the entry point moves up the downstream of the manufacturing process, the stability and responsiveness will decrease.

This inverse relation is because an increase in flexibility allows for the design of novel products, which, on the other hand, will lead to longer design time, adjustments in the available manufacturing system to produce it and testing to guarantee that it performs as expected. Hence, an increase in the flexibility has negative impacts on both the stability and responsiveness of the mass customisation system.

From the discussion about the relation of market segmentation and product families made above, we can view mass customisation as the limit of a market segmentation process by the progressive differentiation of the product family to cope with the heterogeneity within the market. Thus, eventually, every customer will get a product that precisely fits his needs. However, depending on the entry point, that level of segmentation may not be achieved. For example, entry points after the fabrication stage must rely on existing components for the customisation of the products. Therefore, even if there is a great variety of components available, it may not be enough to provide each customer with a product that exactly fits his requirements. In other words, groups of customer will be provided the same product, although their requirements are different. This implies that although customers are being served on an individual basis through a configurator, their needs are not being strictly fulfilled due to limitation on the flexibility of the product family or the customisation process.

As it will become evident in the following chapters, our approach to the customisation of product families is not based on a predefined set of components. Actually, it allows the design of components during the customisation process, which allows the point of customer involvement in the mass customisation system to be moved further to the right in Figure 6, thus providing a lot more flexibility to the mass customization process to specify a product that fits to the needs of the customers. On the other hand, our approach to represent product families has, as one of its pillars, a generic architecture that comprises the architectures of all the members of the product family. It also provides a clear delimitation of the design space within which its members are located. As it will be argued in the conclusion, these conditions will allow for the setting up of a stable mass customization system. Moreover, our approach to the customisation of product families can be fully automated and if properly modelled, can provide assurances that the products been derived perform as expected, thus contributing for the responsiveness of the mass customisation system. Therefore, with our approach, it will be possible to chance the balance between the performance factors of the mass customisation system by increasing its flexibility without compromising its stability and responsiveness.

Chapter 3

Product family modelling

The variability of a product family may be defined within the functional, technological and physical domains [34]. Nevertheless, any functional or technological variation will ultimately appear as a physical variation. Consequently, any two variants within the product family will necessarily have distinct physical models, and the differences will be related either to the components or to their structural organization. Hence, at the physical domain we have a uniform and inclusive account of the variability within a product family.

Because of their variability, no physical model can stand as the representative for all members of the product family. Therefore, in what follows we will develop a generalization of the physical models that is capable of incorporating all the variability within the product family. To define this generic physical model, first we will introduce the concept of component types and the criteria for their classification. To represent the physical models of the members of the product family we will use tree graphs. The root node will stand for the product itself, while all other nodes in the tree will stand for its components, and the hierarchical structure of the tree graph will define the assembling organization of the product. The leaf or terminal nodes are single components or compound components that will not be further decomposed. They are the primitive components of the product family.

3.1 Component types

Let $\mathcal{F} = \{P_1, P_2, \dots, P_n\}$ be a product family and E_i the physical model of product P_i . As noted above, E_i will be represented by a tree graph with the root node standing for P_i and the leaf nodes for the set of

primitive components C_i . Hence, the set of all primitive components for \mathcal{F} is defined by $C = \bigcup_{i=1}^n C_i$.

Each product $P_i \in \mathcal{F}$ realizes a set of overall functions and has a set of attributes. For example, the main functions of an ATS are the monitoring of the utility, starting/stopping the power generator and making the transfer of the load between the utility and the power generator. The number of phases it can monitor and the types of generators it can control are some of its attributes. Assuming that the product family members have modular architectures, to each $x \in C_i$ we can associate a set of sub-functions which, when assembled according to E_i , bring forth the overall functions of P_i . The same rationale is true regarding the attributes.

Primitive components in C can be organized by the following relation. Components $x, y \in C$ are said to be alternative components if and only if they have similar functionality and have been designed to replace each other to create product variety. It should be noted that we did not require alternative components to have the same interface. For example, the enclosures for the ATS product family have two different interfaces. One with a hole to attach the buzzer, in case the customer wants the ATS to monitor the power generator, and the other without such hole.

Being an alternative component is an equivalence relation over C for it is reflexive (every component x is an alternative to itself), symmetrical (if x is an alternative to y then y is an alternative to x) and transitive (if x is an alternative to y and y is an alternative to w then x is an alternative to w). Consequently, this relation partitions C into equivalent classes, such that, within the same class, all the components are alternatives to each other. Hence, set C has a correspondent set C^* such that, for each $x \in C$, there is a corresponding class T to which it belongs, and, for all $T \in C^*$, there is at least one $x \in C$ such that $x \in T$.

Definition 1. Let C be the set of primitive components from which the member of the product family \mathcal{F} are derived. Each class of components $T \in C^*$ is a component type.

Every component type $T \in C^*$ can be classified according to the following criteria. If T contains exactly one component from C , it is called a specific component type. Otherwise, it contains two or more components from C and is called a generic component type. On the other hand, if $T \cap C_i \neq \emptyset$ for all $P_i \in \mathcal{F}$, then T is called a common component type. Otherwise, T is called an optional component type. These classes are not all mutually exclusive, but can be combined into four categories. For example, a generic/common component type represents a class of alternative components such that it contains more than one component from C and all members of the product family have a component from this class in its physical model. The other possible categories are generic/optional,

specific/common and specific/optional. In Figure 7, it is depicted schematically the relation between product family members and component types. The members of the product family, shown at the left column, are derived from the components at the middle column. These components are grouped into classes of alternative components (same colour) forming component types. Their classification, shown at the right column, follows the rules defined above.

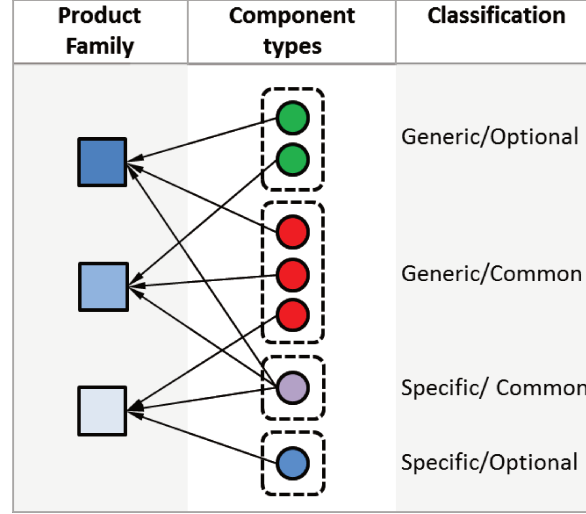


Figure 7. The classification of component types in a product family

In what follows we will introduce a formal definition for the GPS, based on the component types of the product family.

3.2 The generic product structure

Let \mathcal{E} be a tree graph whose root node stands for \mathcal{F} and the leaf nodes stand for the elements in \mathcal{C}^* . We say that the physical structure E_i is isomorphic to a sub graph of \mathcal{E} if and only if the root node in E_i is mapped onto the root node in \mathcal{E} and each leaf node in E_i (a component $c \in C_i$) is mapped onto the corresponding leaf node in \mathcal{E} (the component type $T \in \mathcal{C}^*$ to which c belongs). Moreover, we say that the isomorphism of the set physical structures $E = \{E_i: i = 1, 2, \dots, n\}$, corresponding to the members of the product family, is adherent to \mathcal{E} if and only if the following conditions hold. 1) If T is a common component type in \mathcal{E} , every element in E has a component that maps onto it. 2) If T is an optional component type in \mathcal{E} , at least one element in E (although not all of them) does not have a component that maps onto it.

Definition 2. We say that the structure \mathcal{E} composed of component types from \mathcal{C}^* is a GPS for the

product family \mathcal{F} if and only if the product structure of the members of the product family are isomorphic and coherent to \mathcal{E} .

Note that while each member of the product family may cover just part of the GPS through isomorphism, the property of coherence requires that collectively the members of the product family cover all the GPS. Moreover, this coverage must satisfy the definition of the component types.

To illustrate the concepts of isomorphism and coherence, let $\mathcal{F} = \{E_1, E_2\}$ be a product family and \mathcal{E} a product structure defined out of the components in \mathcal{C} . As depicted in the Figure 8, both E_1 and E_2 are isomorphic to \mathcal{E} . However, because T_{12} is a common component type, and there is no component from E_2 mapping onto it, one of the conditions for coherence fails.

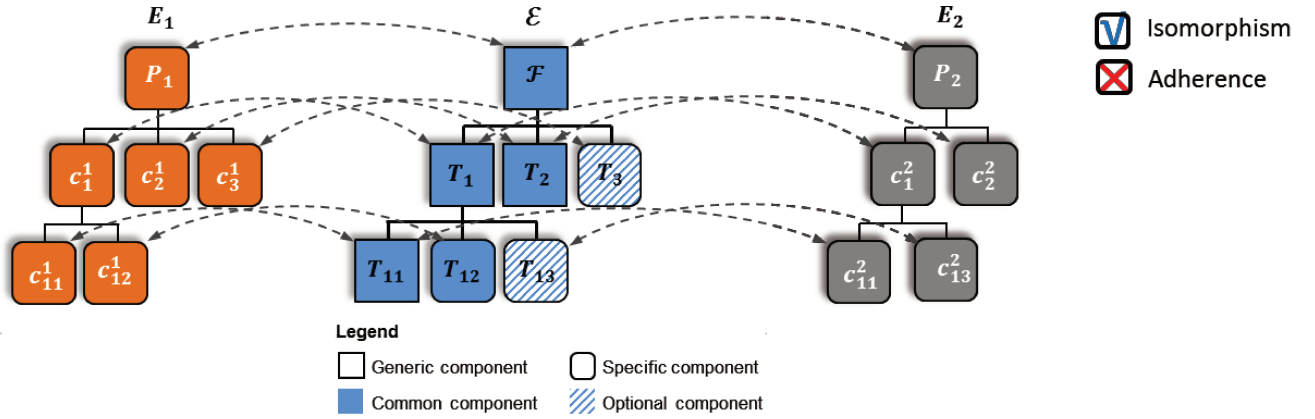


Figure 8. Illustration of isomorphism and adherence

Apparently, the definition of the GPS may seem somewhat restrictive. However, it is possible to construct a GPS satisfying our definition even for a collection of completely distinct products, just by aggregating the root nodes of their corresponding physical models into a common root node. In this case, except for the root node, all other nodes of the resulting GPS would correspond to specific/optional component types. Of course, one would hardly call this collection of products a product family. A more interesting situation arises if some of the leaf nodes in the GPS are common component types.

Note that, based on the classification of the leaf nodes, all the nodes in the GPS can also be classified by applying some rules. For instance, if a leaf node in a GPS is a common (or generic) component type, then all nodes up to the root node are also common (or generic). If a component has daughter nodes that are specific/common and at least one is an optional component type, then the parent node is a generic/common component type, for it admits different configurations of its

subcomponents. Other rules may be more complicated, and sometimes a complete classification may depend on information other than the classification of the daughter nodes. The classification of the component types for the ATS product family can be seen in Figure 9 (see the legend).

From these classification rules, it follows that the leaf component types that are common will define on the GPS a substructure that is common to all members of the product family. However, if some of the component types along this substructure are generic as well, then, besides being a common substructure, it will also vary among the members of the product family. It should be noted that this observation stresses a different perspective from the literature on product families, which tends to regard the common component types as fixed within the product family.

To conclude this subsection, we note that the GPS subsumes the physical structure of all the members of the product family, and works as an envelope for its variability. Hence, it delimits the design space around the members of the product family. However, the GPS alone is not sufficient to determine what physical structures in that delimited design space correspond to a valid product and which one meets a given set of customer requirements. Therefore, in the following subsection, we will introduce a complementary model to the GPS to accomplish those goals.

3.3 The constraint network extended with design functions

A constraint network (CN) model consists of a set of variables, each one associated to a finite set of values, and a set of constraints restricting the values they can take simultaneously. A solution to a CN model is an assignment of values to the variables such that every constraint is satisfied [35].

A CN model is a natural way for constraining the combinations of components on the GPS, with the variables identifying the variations and the constraints describing the couplings that exist between them. The foreground diagram of Figure 9 shows a graphical representation of the CN model for the ATS example. However, in this work, we will introduce an extension to the CN model that is capable of representing procedural knowledge in addition to the declarative knowledge associated to the CN model. This extension will be defined as follows.

Definiton 3. A constraint network model extended with design functions (CN-F) is a triple $\langle V, R, F \rangle$, where V is a set of variables, R is a set of constraints over V and F is a set of design functions. Moreover, for every $x \in V$, there is at least one design function $f_x \in F$ attached to x which generates its values.

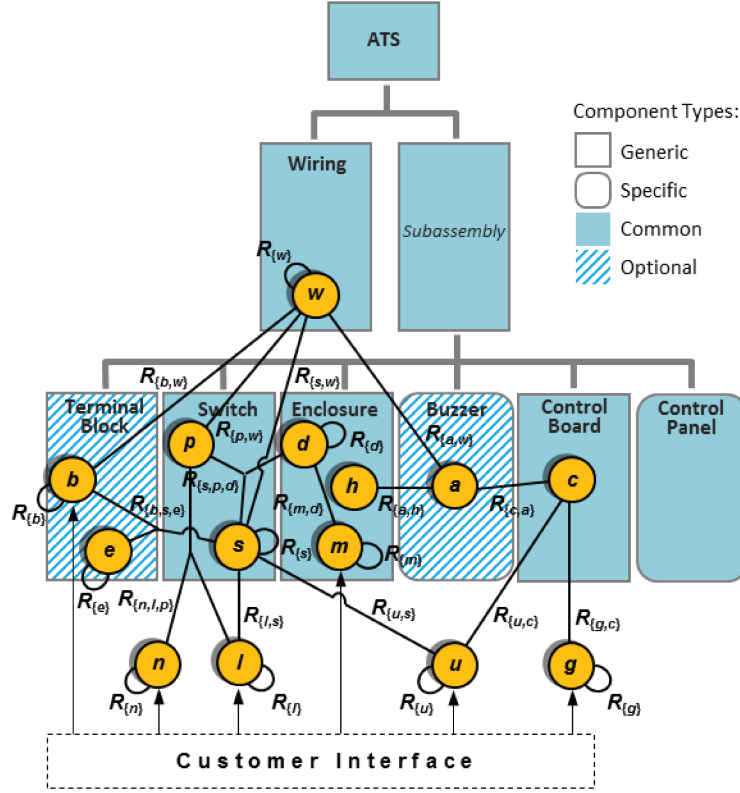


Figure 9. The GPS (background) and the CN-F model (foreground) for the ATS product family

In the following three subsections, we will define each of the elements comprising the CN-F model. In Appendix I, we present the complete description for the ATS product family using this modelling approach. However, in the following chapters we reproduce parts of this modelling to illustrate the concepts introduced.

3.3.1 Variables

Variations among the members of a product family will be identified by variables. They can be mapped on the GPS. However, variations may also be found related to the operational environment of the product family and, therefore, are localized outside the GPS. The set of all these variables will be represent by V . Their scope of variation can extend from a specific attribute to an entire component. In Figure 9, variables are represented as nodes of a network, either on or outside the GPS.

Elements in V are classified primarily as input or output variable. The input variables, represented by the set I , are used to express the customer requirements. In Figure 9, they are indicated by the incoming arrows from the customer interface. Except for the input variable b (which is an inclusion variable associated to an optional component type, namely, the Terminal Block) and the input

variable m (which specifies the material for the component type Enclosure), the remaining input variables u , g , n and l are related to the operational environment of the ATS. The latter input variables specify the attributes of the utility, power generator, neutral wires and load, respectively, to which the ATS will be connected. Every variable on the GPS is an output variable, used to instantiate the GPS into the specific physical models representing members of the product family (see Section 4.2). Output variables are represented by the set O . Since any composite component in the GPS becomes specified after all its subcomponents are specified, to specify a physical model from the GPS it is sufficient to specify the leaf components. Therefore, the output variables are located only on leaf components of the GPS. Except for specific/common component types, which do not vary among members of the product family, every other leaf component type in the GSP must be associated to at least one variable. An optional leaf component type is always associated to an inclusion variable as well, which specifies if the component is to be included or not in the custom product. Therefore, every generic/optional leaf component type has at least two variables; one to specify if the component type is included or not, and all other variables to specify which of the variants of the generic component type is to be used in the custom product.

It should be noted that the sets I and O are not necessarily mutually exclusive, for instance, variable m belong to both. Moreover, we could have introduced intermediate variables, i.e., variables that do not belong to I nor to O , in order to break down a complex relation between variables in V . This type of variable was not necessary in the modelling of the ATS product family.

Every variable has an associated domain of variation that can be defined as the set of all possible variations within the product family in reference to the scope of the variable. In what follows, the domain of variation for variable $x \in V$ is represented by D_x and is defined as the set of all the values that can be assigned to x . We will refer to D_x simply as the domain of x . For example, $D_b = \{0, 1\}$ is the domain of b on the Terminal Block and accounts for its inclusion ($b = 1$) or not ($b = 0$) in the custom ATS. The domain D_s of the composite variable $s = (s_v, s_a, s_d)$ is a bit more complicated. It is defined by Table 2, which lists the voltage s_v , amperage s_a and dimensions s_d of all the available transfer switches. Later (in connection to Figure 11) we will see an example of a more complex variable whose scope extends over the whole component. One important point, which we will call attention here, is that domains are not necessarily defined explicitly in our approach. This issue will be considered in more detail in Subsection 3.3.3 after the design functions have been introduced.

Table 2. Available transfer switches

Item	Amperage	Voltage	Dimensions*
1	25	110	(18,80,90)
2	25	220	(18,80,90)
3	50	110	(19,85,95)
4	50	220	(19,85,95)
5	80	110	(21,95,110)
6	80	220	(21,95,110)

*(w, h, l) in mm

As mentioned above, depending on the value of the inclusion variable, the correspondent component type may not be included in the custom product. However, if this optional component type is also generic, there will be other variables associated to it. To make the value of these variables consistent with the fact that the component is not been included, we can assign to them the value NA (Not Applicable). For example, if $b = 0$, it makes no sense to specify the amperage rate e of the terminal block and, therefore, it $e = \text{NA}$. Hence, for every variable other than the inclusion variable on a generic/optional component type, we will add the value NA to their domains and assign this value whenever the component they are associated with will not be include in the custom product.

To conclude this subsection we will introduce some formal definitions concerning the instantiation of the CN-F model. Let $X = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\} \subseteq V$ be a subset of variables. We define an instantiation of X as the value assignment $x_{i_r} = v_{i_r}$, for $r = 1, 2, \dots, k$, such that $v_{i_r} \in D_{i_r}$. For convenience, an instantiation of X can be represented by a k-tuple $(v_{i_1}, v_{i_2}, \dots, v_{i_k}) \in D_{i_1} \times D_{i_2} \times \dots \times D_{i_k}$. Now, let $Y \subseteq V$ be another subset such that $X \cap Y = \{x_{j_1}, x_{j_2}, \dots, x_{j_l}\}$. We define $(v_{i_1}, v_{i_2}, \dots, v_{i_k})|Y = (v_{j_1}, v_{j_2}, \dots, v_{j_l})$ as the restriction of the instantiation $(v_{i_1}, v_{i_2}, \dots, v_{i_k})$ to the variables that are common to both X and Y . This operation can be generalized to a subset of instantiations $D \subseteq D_{i_1} \times D_{i_2} \times \dots \times D_{i_k}$, so that $D|Y = \{(v_{i_1}, v_{i_2}, \dots, v_{i_k})|Y : (v_{i_1}, v_{i_2}, \dots, v_{i_k}) \in D\}$. The set of all possible instantiations of V is represented by $\mathcal{U} = D_1 \times D_2 \times \dots \times D_n$. This set presents another perspective for the design space of the product family.

3.3.2 Constraints

Variables are related by constraints, which restrict the values that can be assigned to them during the customisation of the product family. Thus, let R_X be a constraint between the set of variables $X = \{x_{i_1}, x_{i_2}, \dots, x_{i_k}\} \subseteq V$ and R the set of all constraints on V . The constraint $R_X \in R$ can be represented by a subset $R_X \subseteq D_{i_1} \times D_{i_2} \times \dots \times D_{i_k}$, which may be defined either intentionally, by logical or mathematical expressions involving the variables, or extensionally, by listing the combinations of values that the variables can take simultaneously. Given an instantiation $(v_{i_1}, v_{i_2}, \dots, v_{i_k})$ of X , if $(v_{i_1}, v_{i_2}, \dots, v_{i_k}) \in R_X$, we say that it satisfies R_X . A solution to the CN-F model is an instantiation of V that satisfies all constraints in R . The set of all such solutions in \mathcal{U} is represented by \mathcal{S} . As we shall argue, this set of solutions corresponds to the members of the product family.

Constraints are classified according to the number of variables involved. $R_{\{s\}} = \{(s_v, s_a, s_d) : (s_v, s_a, s_d) \in \text{Table 2}\}$ is an example of unary constraint. It is used to delimit the domain of variable s , constraining the electrical and dimensional attributes of the transfer switches to those available in Table 2. $R_{\{l,s\}} = \{(l_a, s_a) : s_a \geq l_a\}$ is an example of a binary constraint over the variables s and l , specifically, it constrains the amperage of the switch (s_a) to a value greater than or equal to the amperage of the load (l_a). In the CN-F model of the ATS product family only two of the constraints involve more than two variables. For example, $R_{\{s,p,d\}}$ relates the volume occupied by the switch and the dimensions of the enclosure. However, in the CN-F model constraints may involve any number of variables from V .

It should be noted that constraints relating variables located on different component types define the interfaces between these components. For example, the constraints $R_{\{s,w\}}$ and $R_{\{p,w\}}$ completely define the interface between the component types Switch and Wiring. Together these constraints require that every switch terminal must be connected with a wire with cross-section area compatible to the amperage of the switch.

3.3.3 Design functions

The primary role of a design function is to assign values to the variable to which it is attached (the dependent variable). In general, the values are generated from the values assigned to other variables (the independent variables) related to the dependent variable by constraints. To avoid inconsistencies, the design function is required to incorporate all the constraints involving the dependent and

independent variables related by it.

In preparation to a formal definition of design functions, in what follows we will first define precisely in what sense a function incorporates constraints. Let $R(x) \subseteq R$ be the set of all constraints involving x and $C = \{R_{X_1}, R_{X_2}, \dots, R_{X_t}\}$ a non-empty subset of $R(x)$. Since $x \in X_i$, for $i = 1, 2, \dots, t$, we have that $Y = \bigcup_{i=1}^t X_i - x = \{y_1, y_2, \dots, y_p\}$ is the set of all variables related to x through the constraints in C . Now, let $D^* \subseteq D_{y_1} \times D_{y_2} \times \dots \times D_{y_p} \times D_x$ be a non-empty set of instantiations of $Y \cup \{x\}$ satisfying all the constraints in C and $D = D^*|Y$ the set of those instantiations restricted to Y . By construction, it follows that, for every $(v_1, v_2, \dots, v_p) \in D$, there is at least one value in D_x for which their combination satisfy all the constraints in C . Now, let $f_x: D \rightarrow D_x$ be a function defined on D and with values in D_x . We say that f_x incorporates the set of constraints C if and only if, for every $(v_1, v_2, \dots, v_p) \in D$, $(v_1, v_2, \dots, v_p, f_x(v_1, v_2, \dots, v_p))$ satisfies all constraints in C .

Definition 4. Let $Y = \{y_1, y_2, \dots, y_p\}$ be a set of variables related to x , and $C = \{R_{X_1}, R_{X_2}, \dots, R_{X_t}\}$ the set of all constraints relating the variables $Y \cup \{x\}$ and such that $x \in X_i$, for $i = 1, 2, \dots, t$. If the function $f_x: D \rightarrow D_x$ (defined as above) incorporates all the constraints in C , we say that f_x is a design function for x .

In the remaining of this work, we will refer to design functions in the abbreviated form d -function. Now, let $F(x) = \{f_x^1, f_x^2, \dots, f_x^k\}$ be the set of all d -functions attached to x . The domain D_x will be defined as $D_x = \bigcup_{i=1}^k f_x^i(D^i)$, where D^i is the domain of definition of f_x^i . In other words, D_x consists of all the values that can be generated to x by the d -functions in $F(x)$.

In the modelling of the ATS product family the d -functions are implemented as procedures. They can range from relatively simple procedures to request the customers to make their choices, to more elaborated procedures that can design a component variant. As an example, let us consider the d -function $f_s(u, l)$ shown in Figure 10. It defines the electrical attributes of the transfer switch, represented by the compound variable s . This d -function depends on the attributes of the utility and load, more specifically, on the voltage of the utility u_v and the amperage of the load l_a . The procedure begins equating the voltage of the transfer switch s_v to the voltage of the utility u_v . Then, according to line 2, the procedure collects from Table 2 those switches for which the voltage is equal to s_v and the amperage is greater than or equal to l_a . In the subsequent lines, the procedure determines the amperage s_a and dimensions s_d of the switch. If there is more than one switch satisfying the conditions in line 2, in the next line, the procedure chooses the one with the lowest amperage. Note that, if the value of l_a is

greater than the amperage of all the available switches, then $AMP = \emptyset$ and no value can be assigned to s_a . To avoid this outcome, the values that can be assigned to l_a by the customer are limited to the maximum amperage of the available switches. From the foregoing, it follows that when a value is assigned to s it is unique and the constraints $R_{\{u,s\}}$, $R_{\{l,s\}}$ and $R_{\{s\}}$ are satisfied by the values of the variables involved. Thus, we conclude that $f_s(u, l)$ is a function incorporating these constraints and, therefore, qualifies as a d -function for s . Note that the use of the minimum operator in line 3 also illustrates another important fact about d -functions. Besides constraints, they can incorporate other forms of knowledge. In this particular case, the immediate goal is to optimize the transfer switch; however, this will also lead to the optimization in the dimensions and cost of the entire custom ATS.

Design Function $f_s(u, l)$

Begin

1. $s_v \leftarrow u_v$
2. $AMP \leftarrow \{x : (s_v, x, _) \in \text{Table 2} \wedge y \geq l_a\}$
3. $s_a \leftarrow \min AMP$
4. $s_d \leftarrow z : (s_v, s_a, z) \in \text{Available_Switches}$
5. $s \leftarrow (s_v, s_a, s_d)$

End

Figure 10. An example of design function

As mentioned earlier, a d -function can be used to generate the variants of a component type as a whole. To illustrate this point, let us consider $f_w(a, b, p, s)$ attached to the variable w on the component type Wiring, which comprises the wirings for all members of the ATS product family. A wiring is a set of wires that establishes all the necessary electrical connections between the components of the custom ATS, each wire having the proper cross-section area to support the currents that it will carry. Except for the wires connected to the transfer switch terminals, the cross-section areas of the remaining wires are fixed. For simplicity, we will admit that the lengths of the wires do not vary among the family members.

The procedure to generate the wiring for a custom ATS is based on a master template, illustrated in Figure 11. This diagram contains the wires between every possible terminal for all the members of the ATS product family. The wires connected to the solid terminals in the diagram are common to all members, while those connected to the crosshatched terminals are optional, thick lines indicate multiple wires and the label A is a parameter that specifies the cross-section area of the associated wires. This is defined based on a table relating the cross section area of the available wires

and amperage that are supported by them (see Table I-4 in Appendix I). To derive a custom wire diagram, first all the unnecessary wires are removed from the template in reference to the terminals of the components already allocated to the custom ATS. To do this, it suffices to consider the values of the variables p , b and a , representing the number of switch poles, the inclusion of the terminal block and the buzzer, respectively. After that, the cross-section area of the remaining wires connected to the transfer switch are determined taking into account the rating of the switch, specified by variable s . The resulting diagram is a complete specification of the wiring for the custom ATS.

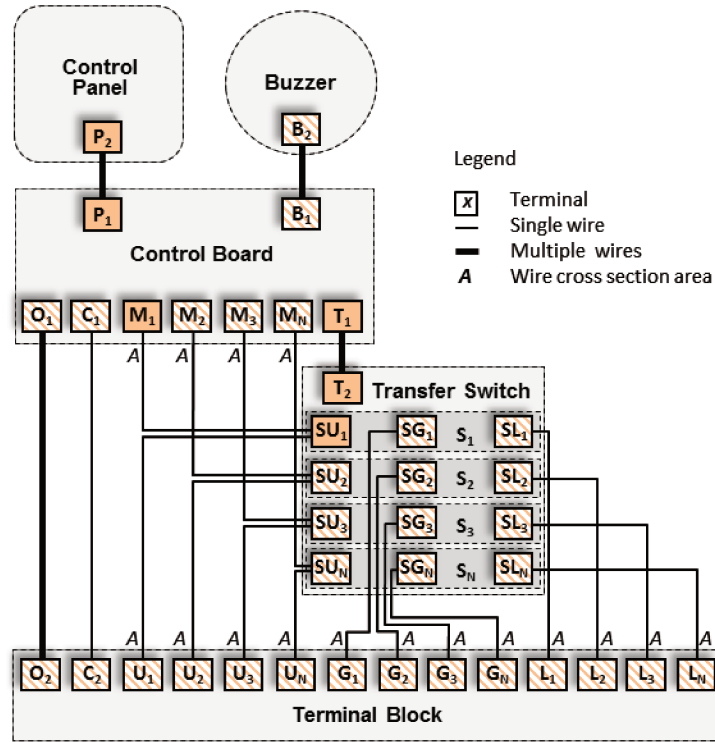


Figure 11. Master template for the wiring of the ATS product family

In general, the customer provides the values for the input variables. Therefore, every variable in I will have a special d -function attached to it, which is called an input d -function. Their role is to help the customer to define the requirements. An input d -function presents the customers a range of options, asking them to choose one (see the dialog windows in Part II of Appendix II). The range of options must be consistent with the domain of the input variable, as defined by the associated unary constraint. For example, because the available switches can deal only with amperages between 1 and 80A, the constraint $R_{\{l\}}$ delimits the amperage of the load l_a to values in that range, and the customer is requested to specify the amperage of the load within that range. In this sense, we can say that the input d -functions incorporate the unary constraints on the input variables to which they are attached. Input

d -functions will be defined by $f_x(y, \mathcal{R}) = x$, where y stands for the customer and \mathcal{R} is the range of options that is presented to him. If the assignment of values to x depends exclusively on the customer, we have that $\mathcal{R} = D_x$. However, as we shall see in Section 4.3, the definition of the range of options may depend on the values that the customer has already assigned to related variables. In this case, the input variable x is also dependent on other variables in V and $\mathcal{R} \subseteq D_x$.

To avoid the introduction of inconsistencies during the value generation process, the d -functions are required to incorporate all the constraints between the dependent and independent variables. However, typically, the variable to which the d -function is attached is not dependent on all the related variables. To illustrate this point, let us consider again the variable s and the d -function $f_s(u, l)$. As it can be verified in Figure 9, s is related to the set of variables $\{u, l, e, w, p, d\}$ by the constraints $R(s) = \{R_{\{s\}}, R_{\{u,s\}}, R_{\{l,s\}}, R_{\{s,e\}}, R_{\{s,w\}}, R_{\{s,p,d\}}\}$. However, through $f_s(u, l)$ the variable s is dependent only on u and l . As it can be verified from the specification of this d -function (Figure 10), the variables u and l and the constraints $R_{\{u,s\}}$, $R_{\{l,s\}}$ and $R_{\{s\}}$ are enough to generate consistent values for s . Making s dependent on any other of those related variables is useless. Actually, the attempt to define the dimensions of the enclosure before knowing the dimensions of the switch and the number of poles would possibly incur into inconsistencies and non-optimal solutions. The d -function $f_w(a, b, p, s)$, described above, gives further evidence of the existence of a logical dependency between variables. Clearly, the specification of the wiring cannot be completed before all the other components inside the enclosure have been defined. However, in this case, the variable w depends on all the variables to which it is related. Hence, we conclude that the set of d -functions F captures the dependency relation that exist between the variables in V , such that, if $f_x(y_1, y_2, \dots, y_p) \in F$, then x depends on y_1, y_2, \dots, y_p . This property is at the core of our instantiation method and will be dealt with in more detail in Subsection 4.1.1.

Before leaving this chapter, we will introduce one more concept. As we have seen, not all constraints in $R(s)$ have been incorporated in f_s . A constraint R_X involving a variable $x \in X$, but not incorporated in f_x , will be said to be free in relation to f_x . However, this constraint may be incorporated in another d -function attached to the same variable or to another variable in X . For example, $R_{\{s,p,d\}}$ is free in relation to $f_s(u, l)$; however, it is incorporated in $f_d(s, p)$ to generate the dimensions of the enclosure. The set of free constraints relative to $f_x \in F$ will be represented by $L(f_x)$.

Chapter 4

Product family customisation process

Mass customisation is about providing the individual customers with products that meet their needs. In our approach, the customisation process takes place in two stages. First, a solution to the CN-F model that is consistent with the customer requirements is found. Then the solution is used to transform the GPS into a specific physical model that corresponds to a member of the product family. It should be noticed that this separation into stages is just a convenience to simplify the customisation process. In principle, they could be carried out in parallel.

It is interesting to note that some general conclusions about the relationship between customer requirements and product family members can be drawn just by taking into account the relationship between the sets I and O . For example, if the set of input variables is not contained in the set of output variables ($I \not\subseteq O$), there may exist solutions $S_1, S_2 \in \mathcal{S}$ such that $S_1|I = S_2|I$ (the solutions restricted to I are equal) while $S_1|O \neq S_2|O$ (the solutions restricted to O are different). In other words, the requirements of the customer may be associated to more than one product family member. In this case, some provision should be made for deriving the best solution according to some criteria. In our approach, this is accomplished through the d -functions, as in $f_s(u, l)$, by the selection of the switch with the minimal amperage of those qualified.

4.1 Finding solutions to the CN-F model

A solution to the CN-F model is an instantiation of V such that no constraint in R is violated. The instantiation process begins with the assignment of values to variables from I by the customer, and

proceeds towards the variables in O guided by the dependencies between the variables established by the d -functions. However, not necessarily all variables in I must be assigned values for the instantiation process to proceed. If the input variable is attached with an alternative d -function, in case the customer does not assign its value it can still be generated from the values of related variables. In Figure 12, we propose an instantiation algorithm to carry out this process. Since it will be instrumental in the demonstration of the conditions for a backtrack-free instantiation process, in this subsection we will examine it in detail. However, first we will introduce some preliminary definitions and assumptions.

Let us suppose that the variable x depends on the value of y for the generation of its value by f_x . Then, if f_y is a d -function attached to y , we can say that f_x depends on f_y . Moreover, if y has more than one d -function attached to it, f_x also depends on each of them alternatively. Otherwise, if x do not depend on other variables in V by means of f_x , we say that f_x is an independent d -function. It should be noted that, although every independent d -function is an input d -function, the other way round is not true. In Section 4.3 we will present an example of an input d -function which depends on other variables to define the range of options that will be presented to the customers. Based on the foregoing discussion, we conclude that the dependency between variables in V induces a dependency relation between d -functions in F . Now, it may happen that following a sequence of dependencies we get back to a previous d -function. More formally, we say that the d -functions f_x, f_y, \dots, f_z form a dependency loop, if each d -function depends on the previous one in this sequence, and f_x also depends on f_z . Next, we will introduce a condition for the ordering of F based on this dependency relation which prevents loops.

Definition 5. Let $F = \{f_1, f_2, \dots, f_p\}$. We say that F is ordered if the d -functions that satisfies the following condition: for every $i = 1, 2, \dots, p$, f_i is either an independent d -function or all the d -functions it depends on precedes it in that order.

One simple procedure to verify if F can be order is given next. First, move all the independent d -functions to the left of the set F . Let us call this the ordered part of F . If all the d -functions in F have been moved, then F satisfies Definition 5 and, consequently, it is ordered. Otherwise, find within the unordered part of F a d -function that depends only on d -functions in the ordered part and move it to the end (right) of the ordered part, thus extending the ordered part. Repeat this process until the unordered part of F is empty. When the unordered part is empty the procedure stops and F is ordered according to Definition 5. To show that this procedure is sufficient to order F in the absence of loops, we demonstrate the following theorem.

Theorem 1. The set F satisfies the ordering condition if and only if there is no dependency loop between the d -functions in F .

If F can be ordered, it follows immediately from Definition 5 that there is no loop between the d -functions. Now, let us suppose that at some point of the procedure described above, the unordered part is not empty and we cannot find a d -function that depends only on d -functions in the ordered part of F . This means that every d -function in the unordered part depends on at least one d -function that is still in the unordered part. In this case, no arrangement of the d -functions in F will satisfy the ordering condition. Next, we will demonstrate that if this condition happens, there is at least one loop in the unordered part.

Let $\tilde{F} = \{f_{i_1}, f_{i_2}, \dots, f_{i_k}\}$ consist of the unordered part of F . By definition, every d -function in \tilde{F} must depend on some other element inside it. Now, going from right to the left and starting with f_{i_k} , move every d -function to the right just behind the first element encountered it depends on. Of course, if this element is already the preceding one in the order of \tilde{F} , no displacement is necessary. Apply the same procedure to every d -function in \tilde{F} until f_{i_1} is reached. However, if a d -function that has already been considered previously is encountered, skip that element and go to the next one to the left. When this process reaches f_{i_1} , the elements in \tilde{F} are arranged so that all elements to the right of f_{i_1} depends on it directly or indirectly, through a chain of dependencies. From the definition of \tilde{F} , it follows that f_{i_1} also depends on some other d -function to its right in \tilde{F} , thus defining a loop. ■

Note that, if F can be ordered to satisfy Definition 5, then there may be alternative arrangements of d -functions in F that also satisfy that ordering condition.

Assumption 1. In what follows, we will assume that F can be order.

According to the procedure described in Figure 12, the instantiation algorithm uses the set G to collect the variables whose values are being assigned, and the set L to collect free constraints that have not been verified up to a given point in the execution of the algorithm. We say that a d -function is enabled during the instantiation process if and only if the variables it depends on are contained in G . Since the sets G and L are empty at the beginning of the instantiation process, the independent d -functions are the only functions that are enabled. If a variable has been assigned the value NA, every constraint in set L involving this variable is trivially satisfied. This is justified by the fact that the correspondent optional component type will not be included in the custom product. In what follows, we will examine the steps of the instantiation algorithm.

Let us begin with the steps of the algorithm where the conditions leading to a failure during the instantiation process are located. This happens at step 7, if the selected d -function fails to generate a value for the variable under consideration and there is no alternative d -function attached to it. More specifically, a d -function $f_x: D \rightarrow D_x$ will fail to generate a value if the combination of values of the variable it depends on (all of them already in G) fall outside the domain of definition D . The failure at step 12 happens if a free constraint in L is not satisfied by the values of the variables in G . We will refer to those failures as inconsistencies of type 1 and 2, respectively.

The instantiation algorithm:

Begin

1. Following the order in F , remove the first f which is enabled
2. If f successfully assigns a value to x
3. Then add x to G , remove from F all alternative functions attached to x
4. and make $L = L \cup L(f)$
5. Else If there is another generative function attached to x
6. Then go to step 1
7. Else go to End and return "Failure"
8. For every $R_X \in L$
9. If $X \subseteq G$
10. Then If R_X is satisfied
11. Then remove it from L
12. Else go to End and return "Failure"
13. If $F \neq \emptyset$
14. Then go to step 1
15. Else go to End and return "Success"

End

Figure 12. The instantiation algorithm

To illustrate inconsistency of type 1, let us consider the d -function $f_s(u, l)$, defined in Figure 10. If the customer assigns to the load l an amperage that is greater than the amperages of the switches in Table 2, $f_s(u, l)$ is not defined and an inconsistency will arise. However, this source of inconsistency can be avoided by just making the unary constraint $R_{\{l\}}$ consistent with the available switches. With regard to inconsistency of type 2, the only critical point in our modelling of the ATS product family is related to constraint $R_{\{d, m\}}$, which is free in relation to both f_m and $f_d(s, p)$. In this case, it may happen that the choice of material made by the customer by means of f_m and the dimension of the enclosure assigned by $f_d(s, p)$ do not satisfy the constraint $R_{\{d, m\}}$ (which is defined based on Table 3) leading to an inconsistency. In the next subsections, we will discuss how to avoid this source of inconsistency.

If the execution reaches step 13 and $F = \emptyset$, then all points in V have been instantiated and a solution has been found. To prove this statement we note that, if $F = \emptyset$, then every variable must have been assigned a value (through step 2). Moreover, since every constraint in R was been verified either as a constraint incorporated in a d -function, at step 2, or as a free constraint in L , at step 8, we conclude that no constraint in R has been violated.

Now, we will prove that the algorithm always succeeds at step 1. It is clear that while not all the independent d -functions have been removed from F there will be at least one d -function that is enabled. Thus, let us suppose that all independent d -functions have been removed from F and we still have $F \neq \emptyset$. Moreover, let us suppose by absurd that in the next iteration through step 1 no d -function is enabled. From the assumption on the ordering of F and from the fact that an inconsistency of type 1 did not happened up to this point, it follows that the first d -function not enabled (in the order of the remaining d -functions in F) will depend on variables that have already been assigned their respective values and, consequently, must be in G . Since this conclusion contradicts the initial assumption that no d -function is enabled, this assumption is unwarranted and we can conclude that the algorithm will always succeed at step 1.

At step 2 an additional variable is assigned its value, thus extending a partial solution. However, if the resulting set of values does not match any solution in \mathcal{S} , an inconsistency of type 1 or 2 will necessarily occurred as the execution of the algorithm proceeds. In particular, this may happen while the values to the variables in I are being assigned. If the value assignment of the input variables does not match any partial solution in $\mathcal{S}|I$, we say that there is an inconsistency embedded in the input.

If F can be ordered, no matter how the d -functions are actually arranged in F , the instantiation algorithm introduced above can be executed. Nevertheless, it is unable to deal with backtracking for the removal of inconsistencies or with iteration due to the tear up of dependency loops (an issue that will be discussed further in the next section). In spite of these restrictions, it is sufficient for our goals in this work.

4.1.1 Instantiation patterns

Every time the instantiation algorithm is run, a subset of the d -functions in F is executed in sequence until an inconsistency arises or a solution for the CN-F model is found. This instantiation process can be represented as the construction of a dependency graph composed of nodes, representing the variables, which are added to the graph as their values are generated, and directed arcs, that represent the dependencies between the variables as established by the d -functions that has been used to generate

their values. Such a graph can be organized into dependency levels. Level 0 is composed of input variables which depends exclusively on the customer for the assignment of their values. Level n (for $n > 0$) is composed of variables that depend on at least one variable at level $n - 1$. As we shall see below, if different sequences of d -functions are executed, probably different instantiation graphs will arise. Figure 13 illustrates the two instantiation graphs, labelled G_1 and G_2 , for the CN-F model of the ATS example. Nodes with a double circle represent output variables. The dashed curve around nodes m and d in G_1 represents constraint $R_{\{d,m\}}$, which is free in relation to both f_m and $f_d(s,p)$ in that graph.

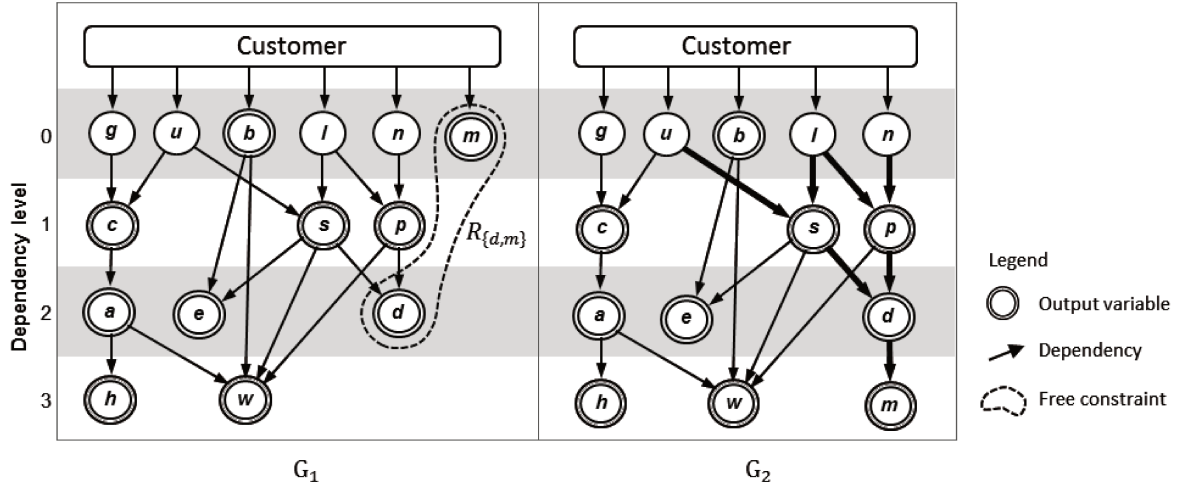


Figure 13. Instantiation graphs G_1 and G_2 for the ATS example

Instead of variables, the nodes in the instantiation graph can be labelled with the d -functions that were used to generate their values, giving rise to a correspondent structure of dependencies between the d -functions. Since such graphs are patterns of functions executed during the instantiation process that will repeat for many different sets of inputs, we will refer to them as an instantiation patterns. Note that, by construction, the set of d -functions that corresponds to an instantiation pattern satisfies the ordering condition.

Definition 6. An instantiation pattern is a subset $P \subseteq F$ satisfying the ordering condition and such that, for every $x \in V$, there is only one $f_x \in P$.

The advantage of defining instantiation patterns in terms of d -functions instead of variables is that their representation are unique, i.e., for each $P \subseteq F$ that satisfies Definition 6 there is only one graphical representation associated to it. In contrast, the same instantiation graph may be the result of distinct instantiation patterns in F . As we will show in Appendix I, in the ATS example there are only

two instantiation patterns P_1 and P_2 , which are associated to the instantiation graphs G_1 and G_2 , respectively. As it will be proved next, if F satisfies the ordering condition, then it is always possible to find an instantiation pattern for the CN-F model.

Theorem 2. If F satisfies the ordering condition, then it contains at least one instantiation pattern P .

To prove this statement, let P be an empty set of d -functions and $V^* = V$. While V^* is not empty, do the following procedure. Remove x from V^* , find any $f_x \in F$ and move it to P . If f_x is not an independent d -function, for each of the variables that x depends on through f_x , first, verify if there is already a d -function attached to it in P . If this is the case, go to the next variable. Otherwise, remove it from V^* and move one d -function attached to it from F to P . Because F satisfies the ordering condition, we can continue expanding the dependency tree to which x is a root until it is completed, that is to say, ending up with independent d -functions. Since V is a finite set, eventually V^* will become empty and this process will terminate. By construction, the resulting set P satisfies the ordering condition and, for every $x \in V$, there is only one $f_x \in P$. Based on the foregoing, we conclude that P satisfies the conditions of Definition 6. ■

From the procedure described above it follows that, if a variable is associated to more than one d -function, there will exist more than one instantiation pattern in F .

Because of our assumption on F , all variables in the instantiation graph are connected to variables at level 0 through dependencies. Hence, tracing back all the possible paths from a given node x towards the nodes at level 0, we determine a dependency tree graph that has x as its root and the nodes at level 0 as its leaves. However, as for instantiation patterns, dependency trees will also be defined in terms of d -functions.

Definition 7. Let $T(f_{i_k}) = \{f_{i_1}, f_{i_2}, \dots, f_{i_k}\}$ be an ordered non-empty subset of F . $T(f_{i_k})$ is a dependency tree in F if and only if it satisfies the following conditions. If f_{i_k} is an independent d -function attached to x , then $k = 1$. Otherwise, for $1 \leq j \leq k$, all variables to which f_{i_j} depends on has one and only one d -function attached to them in $T(f_{i_k})$ and these d -functions precede f_{i_j} in the sequence defined by $T(f_{i_k})$. Moreover, for every $j < k$, f_{i_j} is attached to a variable which another subsequent d -function in the order of $T(f_{i_k})$ depends on.

From this definition, it follows that a dependency tree $T(f_{i_k})$ is a directed acyclic graph that has f_{i_k} as its root and input d -functions as leaves. For example, the dependency tree $T(f_m) = \{f_m\}$ is

represented by a graph that consists of a single node, while the dependency tree $T(f_m(d)) = \{f_l, f_n, f_s(l), f_p(n, l), f_d(s, p), f_m(d)\}$ is represented by a graph as the one highlighted by heavy arrows in G_2 , with the variables substituted by the correspondent d -functions.

As can be recognized from the instantiation graphs in Figure 13, every d -function in F can be associated to a dependency tree. The main difference in the topology between dependency trees and instantiation patterns is that the latter admits multiple roots. For example, the d -functions attached to h , w , e , d and m in G_1 are roots in the correspondent instantiation pattern P_1 , each one associated to its own dependency tree. Actually, instantiation patterns can be regarded as the composition of dependency trees.

Theorem 3. Let \mathcal{T} be the set of all dependency trees in F and \mathcal{T}' a subset of \mathcal{T} . If the set $P = \bigcup_{T \in \mathcal{T}'} T$ is such that, for every $x \in V$, there is only one d -function attached to x , then P is an instantiation pattern.

By definition, for every $x \in V$, there is only one $f_x \in P$. Thus, to prove the theorem it is necessary to show that P satisfies the ordering condition. It can be easily demonstrated that the union of subsets of F satisfying the ordering condition has the same property. Therefore, since all $T \in \mathcal{T}'$ satisfy the ordering condition, it follows that $P = \bigcup_{T \in \mathcal{T}'} T$ also satisfies the ordering condition. ■

It follows that every instantiation pattern P is composed by a set of dependency trees $\mathcal{T}' \subseteq \mathcal{T}$. Next, we will prove that this composition in terms of dependency trees is unique. By definition, we know that for every $x \in V$, there is only one $f_x \in P$. However, it remains to be proved that there can be only one dependency tree in P to which f_x is the root.

Theorem 4. Let $T_1(f_x), T_2(f_x) \in \mathcal{T}$ be two distinct dependency trees that has f_x as their root and P an instantiation pattern. Then it cannot be the case that they belong to the same instantiation pattern simultaneously, that is to say, either $T_1(f_x) \subseteq P$ or $T_2(f_x) \subseteq P$, but not both.

Assuming that $T_1(f_x) \neq T_2(f_x)$, it must be the case that, while tracing back the dependencies in synchrony from their common root, at some point we will find two distinct d -functions associated to the same variable, for otherwise there would be no difference between them contrary to our assumption. But since alternative d -functions cannot belong to the same instantiation pattern P , $T_1(f_x)$ and $T_2(f_x)$ cannot both belong to P . ■

If we define a maximal tree as a dependency tree that is not strictly contained in any other dependency tree in F , it follows that any instantiation pattern P can also be expressed in terms of the

maximal trees. For example, the instantiation pattern P_1 can be expressed as the composition of five maximal dependency trees, each one associated to one of the roots on it. Such a composition corresponds to a minimal arrangement, that is to say, any other composition of dependency trees from \mathcal{T} to express the instantiation pattern P will require more elements. Thus, once the maximal trees in F has been defined, the instantiation patterns in F can be determine by combining the maximal dependency trees according to the conditions of Theorem 3. However, we will not explore further this point in this work. As we shall see below, the concept of dependency tree is particularly useful in connection to inconsistency removal during the instantiation process to define the scope of change of the values of the input variables.

In the literature, dependencies between variables are analysed using the Design Structure Matrix (DSM) technique [36]. The rows and columns in a DSM correspond to the variables in V and the marks inside the matrix to dependencies between variables. Then, loops between variables correspond to couplings in the DSM. Couplings are a common outcome in the design of products, and product families by extension. On the other hand, from Theorem 1, it follows that the ordering assumption on F rules out dependency loop between d -functions and, consequently, between variables as well. However, in practice loops tend to be confined within the modules of the product architecture [37]. In our approach, loops can be confined within the d -functions by grouping variables that are highly coupled and attaching the d -function to the group of variables. Therefore, we can conclude that our assumption on the ordering of F is not much restrictive in this respect. Anyway, couplings can be torn up by transforming one of the variables forming a loop into a special input variable. However, this would require the assignment of estimated values to these input variables at the beginning of the instantiation process and, possibly, subsequent refinements, into an iterative process, a capability that the algorithm described in Figure 12 does not have.

Since loops are ruled out by the ordering assumption on F , the instantiation algorithm does not have to deal with them. However, it is able to deal with multiple instantiation patterns in F . Because alternative d -functions are discarded at step 2 of the instantiation algorithm, only one of the patterns in F will be followed during the instantiation process. Which one is actually followed will depend on the specific order of the d -functions in F , and on the inputs from the customer. In the ATS example, the d -functions f_m and $f_m(d)$ are attached to the same variable, and each one is associated to a different pattern (patterns P_1 and P_2 , respectively). If f_m is placed before $f_m(d)$ in F , the preference to assign the value for the variable m is given to the customer. If the customer effectively assigns a value to m , then the instantiation process follows pattern P_1 , otherwise it follows pattern P_2 . On the other hand, if f_m is

placed after $f_m(d)$, then the instantiation will follow pattern P_2 and the customer will never be asked to provide a value for m because $f_m(d)$ always succeeds. Actually, in this case, f_m could be removed from F with no further consequence, since only pattern P_2 would be followed for all the inputs. The main advantage is that an inconsistency of type 2 will never happen. However, as discussed in Section 4.3, the penalty for this particular modelling of the ATS product family is that the customer will not be asked to make a choice regarding the material of the enclosure and some solutions may be lost. In that section, we will also see how to overcome these negative effects by substituting f_m by another input d -function.

From the foregoing discussion, we conclude that it is possible to control the interactions of the customer in the customisation process by the order of the d -functions in F . Moreover, some inconsistencies may be prevented by the appropriate adjustment of the range of options for the input variables. However, it may be difficult to prevent all the inconsistencies embedded in the input at the outset, since inconsistencies may reveal themselves only at a deeper level of the instantiation process.

Within a dependency tree, value assignment follows a strict order. Therefore, to remove an inconsistency during the instantiation process, it is necessarily to go back to the input variables to change their values. Fortunately, the scope of change can be quite restricted, taking into account only the input variables in the scope of the dependency tree of the variables involved in the inconsistency. To define this scope of change precisely, let us define $I(x, P)$ as the set of input variables in the dependency tree $T(f_x) \subseteq P$. If the inconsistency is of type 2, then it must be the case that a constraint $R_{\{x, y, \dots, z\}}$ has been violated by the combination of the values assigned to the variables in $\{x, y, \dots, z\}$. In the attempt to remove the inconsistency, the values of at least one of those variables must be changed. However, given that these changes must be effected by the input d -functions associated to those variables, the set $I(x, P) \cup I(y, P) \cup \dots \cup I(z, P)$ defines the scope of the inconsistency in relation to the instantiation pattern P that is being followed. For example, let us suppose that an inconsistency of type 2 arises by means of the free constraint $R_{\{m, d\}}$ in instantiation pattern P_1 after the value for d is generated. To remove this inconsistency, we should consider changing only the values of the input variables l, n and m . If the inconsistency is of Type 1, then a d -function $f_w(x, y, \dots, z)$ is not defined for the combination of values assigned to the independent variables $\{x, y, \dots, z\}$. In this case we would be in a situation similar to the previous one, having to change the values of some of the input variables in the set $I(x, P) \cup I(y, P) \cup \dots \cup I(z, P)$. Because values at the dependency level 0 correspond to customer requirements, any value change at this level should be carried out by the customer, as part of his interaction in the customisation process. After some change is made, only the affected dependency

trees must have their values generated again. However, this may lead to an iterative process that goes on until all inconsistencies embedded in the input have been removed.

In this work, we will not pursue further the procedures for removing inconsistency. Instead, in the next subsection, we will define two modelling conditions for which inconsistencies do not occur.

4.1.2 *Conditions for consistency*

Up to this point, we have demonstrated that if there is an inconsistency embedded in the customer input, it will emerge during the instantiation process. Now, let us suppose that the values assigned to the variables in I by the customer belong to $\mathcal{S}|I$, i.e., the input is part of a solution. Given that d -functions generate values that are locally consistent, one may wonder if a partial solution can be expanded into a solution for the CN-F model without incurring into inconsistencies. However, local consistency alone does not guarantee that a partial solution can be expanded into a complete solution without backtracking [35]. Based on the analysis of the causes of failure of the instantiation algorithm, in what follows we will introduce two consistency conditions to the CN-F model such that the instantiation algorithm will be able to execute without failing.

Consistency Condition 1. For every $x \in V$, there is at least one $f_x \in F$ which is defined for every instantiation of the variables it depends on.

To understand the consequences of this consistency condition, let us begin reconsidering the definition of d -functions. Since a d -function incorporates a subset of constraints from R , some combinations of values for the variables it depends on may not satisfy all those constraints and, therefore, the d -function will be undefined for those values. If this is the case, it implies that the domain of definition of the d -function is a strict subset of the set of all possible combinations of values for the variables it depends on. Formally, this is expressed by $D = D^*|Y \subset D_{y_1} \times D_{y_2} \times \dots \times D_{y_p}$. Therefore, in general, a d -function may not satisfy Consistency Condition 1. On the other hand, if the domain for f_x satisfies the condition $D = D^*|Y = D_{y_1} \times D_{y_2} \times \dots \times D_{y_p}$, it will be defined for every instantiation of the variables it depends on. However, to satisfy this condition does not imply that the d -function is trivial. For example, the d -function for generating the wiring for the custom ATS is defined for every combination of values it depends on. These values are used in a complementary way to determine the wires from the template that will be kept and their cross-section areas. In our modelling of the CN-F for the ATS, all d -functions satisfy the first consistency condition. This is shown in the Appendix I.

Consistency Condition 1 can also be achieved by restricting the values that can be assigned to the variables the d -function depends on, so that the combinations of their values are always within its domain of definition. For example, by delimiting appropriately the domains of the amperage of the load l and the voltage of the utility u to amperages and voltages that can be handled by the available switches, we can ensure that the combination of their values are within the domain of definition of $f_s(u, l)$. In our modelling of the ATS product family (Appendix I), this restriction was effected through the constraints $R_{\{l\}}$ and $R_{\{u\}}$. Eventually, restricting the unary constraints on the input variables that a f_x depends on is not enough and it will be necessary to introduce new constraints. In this case, some of the d -functions involved must be redefined to incorporate the new constraints.

Theorem 5. If the CN-F model satisfies the Consistency Condition 1, no inconsistency of type 1 will occur during the instantiation process.

To prove this theorem, we simply note that, in the worst case, the instantiation algorithm will iterate between steps 2 and 1 until an enabled f_x that satisfies Consistency Condition 1 is reached. This implies that, during the instantiation process, every $x \in V$, at least one d -function attached to x will succeed assigning a value to it. Therefore, we conclude that an inconsistency of type 1 will never occur at step 2 under Consistency Condition 1. ■

Next, we will prove that if the CN-F model satisfies Consistency Condition 1, it implies that it contains at least one instantiation pattern $P \subseteq F$ that also satisfies this condition.

Theorem 6. The CN-F model satisfies the Consistency Condition 1 if and only if it contains at least one instantiation pattern that satisfies this condition.

If there exists an instantiation pattern that consists of d -functions that are defined for every instantiation of the variables they depend on, it is obvious that the CN-F model also satisfies the Consistency Condition 1. Now, for every $x \in V$, let us remove all the d -functions attached to it from F , except one which is defined for every instantiation of the variables it depends on. Let us call the remaining set of d -functions P . Hence, for every $f_x \in P$, we can also find in P d -functions attached to the variables it depends on. Moreover, there is no loop between the elements in P , otherwise this loop would also belong to F , which by assumption does not have loops. Therefore, we conclude that P is an instantiation pattern for the CN-F model. ■

Based on Theorem 6, we can rephrase Theorem 5 as follows. If the CN-F model contains at least one instantiation pattern that satisfies the Consistency Condition 1, no inconsistency of type 1 will occur during the instantiation process. Next, we will introduce our second consistency condition.

Consistency Condition 2. Let $P \subseteq F$ be an instantiation pattern. Every constraint in R is incorporated by some d -function belonging to P .

The next theorem proves that this condition is enough to avoid inconsistencies of type 2.

Theorem 7. If every instantiation pattern $P \subseteq F$ satisfies the Consistency Condition 2, no inconsistency of type 2 will occur during the instantiation process.

To prove this statement, let us assume by contradiction that after $f_x^i \in F$ have assigned a value to x , the constrain $R_X \in L$ involving the variables $X = \{x, y, \dots, z\}$ has been violated. Since the value assignment to x was made only by the i -th d -function in F , we can conclude that R_X is not incorporated in any of the preceding d -function executed by the instantiation algorithm, otherwise they would not be enabled. From the assumption that every constraint in R is incorporated by some d -function in P and from the fact that R_X must be incorporated by a d -function attached to one variable in X , we can conclude that R_X must be incorporated by a d -function attached to x . But, since x can have at most one d -function attached to it in \mathcal{P} , it follows that R_X must be incorporated in f_x^i . This implies that the value assigned to x must be consistent with the values of the variables in $X - \{x\}$ with respect to R_X . However, this leads to a contradiction with our assumption at the beginning of the argument, thus proving our proposition. ■

Since the constraint $R_{\{m,d\}}$ is not incorporated by f_m nor $f_d(s,p)$ in the instantiation pattern P_1 of the CN-F model for the ATS, it follows that it does not satisfies the Consistency Condition 2. Actually, as we have discussed above, inconsistencies may arise in connection to the variables m and d during the instantiation process following this instantiation pattern.

From Theorems 6 and 7, we conclude that, if the CN-F model has at least one instantiation pattern that satisfies the Consistency Condition 1 and all of them satisfy the Consistency Condition 2, then inconsistencies of type 1 and 2 will not occur during the execution of the instantiation algorithm (Figure 12). Consequently, for all the inputs belonging to $\mathcal{S}|I$, a solution will be found without backtracking. However, it is interesting to note that under those conditions, actually all inputs in $\mathcal{U}|I$ can be extended to a solution without backtracking.

Theorem 8. If the CN-F model has at least one instantiation pattern that satisfies the Consistency Condition 1 and all of them satisfy the Consistency Condition 2, then $\mathcal{S}|I = \mathcal{U}|I$.

To prove this theorem, we begin noting that by definition $\mathcal{S}|I \subseteq \mathcal{U}|I$. Now, let us suppose that $\mathcal{S}|I \neq \mathcal{U}|I$, i.e., there is at least one assignment of values to the input variables I which is not part of any solution in \mathcal{S} . If this input is not part of a solution, it implies that an inconsistency will arise during the execution of the instantiation algorithm, either at step 2 or 3. However, we have already proved that under the conditions of the present theorem this cannot happen. Therefore, it follows that $\mathcal{U}|I = \mathcal{S}|I$. ■

In Subsection 4.3, we will discuss an implication of this fact for the customisation process. Next, we will consider how solutions are used to transform the GPS into a specific physical model, thereby establishing the correspondence between the set of solution \mathcal{S} and the product family \mathcal{F} .

4.2 Transforming the GPS into physical models

Solutions in \mathcal{S} are instantiations for the variables in V and, in particular, for the subset O . Given a solution, the initial stage in the transformation of the GPS into a specific physical model consists of removing the components types not required for the custom product. This is a bottom-up process, which starts with the removing of every optional leaf component from the GSP for which the corresponding inclusion variable has been assigned a value that prescribes its removal. In case a compound component type has all its sub-components removed, the component itself is removed. After the GPS has been stripped of the components not required by the solution, the second stage of the transformation process is the substitution of the generic components by specific variants chosen from the corresponding classes of components. Once again, this is a bottom-up process. However, after the generic leaf component types have been substituted, a specific physical model will immediately emerge from the GSP. By means of this transformation process every solution in \mathcal{S} derives a specific physical model.

By construction, the resulting physical model is isomorphic to the GPS of the product family and is coherent to its component types. Since the choice of the components was made by a solution of the CN-F model, it implies that no design constraint of the product family introduced in the CN-F model is violated. Now, if every relevant design constraint has been elicited and introduced in the CN-F model, we can conclude that every solution in \mathcal{S} corresponds to a member of the product family. In this case, we say that the CN-F model is representative of the product family. Moreover, if $S_1, S_2 \in \mathcal{S}$ are any two solutions such that $S_1|O \neq S_2|O$, the physical models they instantiate are also distinct. Thus,

the correspondence between solutions and physical models is such that distinct solutions in relation to the output variables lead to distinct members of the product family.

4.3 Customisation of the ATS product family

In this section we will illustrate the customization process introduced in the preceding sections, and will discuss some of the consequences of turning the customisation of the ATS product family into a backtrack-free process. As detailed in Appendix I, the set of d -functions for the ATS is defined by $F = \{ f_u, f_g, f_l, f_n, f_b, f_m, f_s(u, l), f_p(n, l), f_c(u, g), f_e(b, s), f_m(d), f_d(s, p), f_a(c), f_h(a), f_w(a, b, p, s) \}$, which contains the instantiation patterns P_1 and P_2 (see Table I-5 for their definition). In Appendix I, we also demonstrate that both patterns satisfy the Consistency Condition 1, but the instantiation pattern P_1 does not satisfy the Consistency Condition 2 because the constraint $R_{\{d,m\}}$ is not incorporated in any of its d -functions. Thus, the instantiation pattern P_1 is a source for inconsistencies. One option to fix this problem is to remove f_m from F . Then, the resulting set F' coincides with the instantiation pattern P_2 .

Table 3. Available enclosures

Item	Dimensions*	Material
1	(260, 190, 450)	metal
2	(260, 190, 450)	plastic
3	(280, 200, 480)	metal
4	(280, 200, 480)	plastic
5	(310, 210, 530)	metal
6	(310, 210, 530)	plastic
7	(350, 220, 600)	metal

*(w, h, l) in mm

Not having included f_m in F' , it implies that m is no longer an input variable. An immediate consequence is that the customers are prevented from expressing their preferences concerning the material of the enclosure. Another consequence is that the number of possible solutions in the ATS example is reduced. While under pattern P_1 there are solutions for which the material of the enclosure is plastic, under pattern P_2 the enclosure will always be specified as metal. Actually, this happens only

because in our implementation of the d -function $f_m(d)$ we simply select the first option of material encountered in Table 3 for a given dimension of the enclosure. Therefore, it should not be concluded from our example that the elimination of a pattern necessarily leads to the loss of solutions. Moreover, as the following discussions shows, there is another option to eliminate the cause for inconsistencies that has no such side effect.

Suppose that, instead of $f_m(y, D_m) = m$, which presents to the customer y all the values in D_m as options he can choose from, we had an input d -function f'_m that presents only the material from Table 3 that are consistent with the value he already assigned to variable d . In this case, the range of options \mathcal{R} would be a function of d , let us say, $g(d) = \mathcal{R}$, and the input d -function f'_m would be represented by $f'_m(y, g(d)) = m$. This new input d -function would give rise to the instantiation pattern P_3 associated to the instantiation graph G_3 shown in Figure 14. As in the case of the instantiation graph G_2 (Figure 13), the variable m appears at the dependency level 3. However, instead of being automatically generated from the value assigned to variable d , the value for m is again assigned by the customer. Since f'_m incorporates the constraint $R_{\{d,m\}}$ by means of $g(d)$, the pattern P_3 satisfies both consistency conditions. Consequently, we would have a backtrack-free instantiation pattern in which the customer can still choose the material of the enclosure and no solution is lost. Although this change fixes the cause for inconsistency in pattern P_1 , in what follows we will proceed considering the set F' , defined at the beginning of this section.

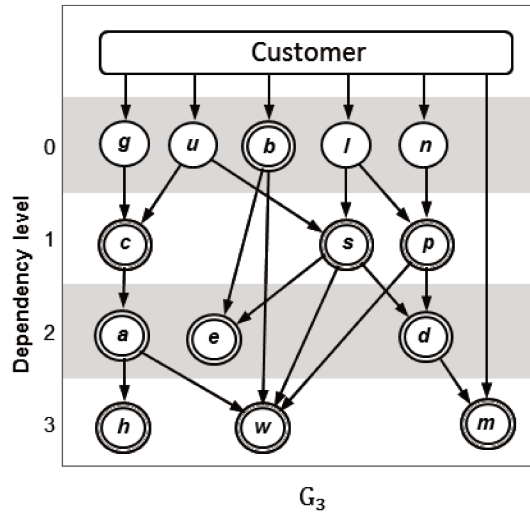


Figure 14. The instantiation graph associated to pattern P_3

Table 4 defines the domains for the input variables (note that m is no longer an input variable). As it can be verified, there are 245,760 possible combinations of values in $\mathcal{U}|I$. This huge number of

inputs is because two of the domains are relatively large; D_l has 240 options and D_g has 32. Nevertheless, not all combinations of values will occur in practice. For example, if the utility is single phase ($u_p = 1$), then it is not expected that the load will have four wires ($l_w = 4$). To prevent such combinations, we could have included additional constraints over the input variables in the CN-F model. However, since these constraints are related to the consistency of the wiring in the application and not specifically to the ATS, we choose not to do so. Anyway, because under the consistency conditions we have that $\mathcal{U}|I = \mathcal{S}|I$, a solution will be found for each of those inputs. For example, for the “inconsistent” input mentioned above, there is a solution that specifies a transfer switch with more poles than needed. A similar outcome will result if the customer informs incorrectly that the neutral and ground are grounded. Thus, although such inputs lead to a solution, the resulting custom product is not necessarily what the customer needs, something that should be expected from an ill-formulated set of requirements.

Table 4. The input variables and their respective domains

Input variable	Domain
u	$D_u = \{(u_v, u_f, u_p) : u_v \in \{110, 220\}, u_f \in \{50, 60\}, u_p \in \{1, 3\}\}$
l	$D_l = \{(l_a, l_w) : 1 \leq l_a \leq 80, l_w \in \{2, 3, 4\}\}$
g	$D_g = \{(g_w, g_m) : 0 \leq g_w \leq 15, g_m \in \{0, 1\}\}$
n	$D_n = \{n : n \in \{0, 1\}\}$
b	$D_b = \{b : b \in \{0, 1\}\}$

Taking into account the range of values of the d -functions attached to the output variables, the set \mathcal{S} contains 9,216 solutions, each one corresponding to a member of the ATS product family. This number of solutions is obtained by multiplying the number of configurations of the 8-position DIP switch (256 options), the configuration of the transfer switch (36 options) and the inclusion or not of the terminal block (2 options), and dividing this product by 2 to account for the fact that the voltage of the transfer switch is already counted in the configuration of the 8-position DIP switch. Since the configuration of all the other components in the ATS are derived from them, they will not add to the number of solutions in \mathcal{S} . Note that this number is 3/80 of the number of inputs. The reason is that, in our example, the transfer switch has only three options of amperage (see Table 1) to deal with 80 possible inputs for the load amperage.

In the remaining of this subsection, we will go through a simulation of the instantiation process of the ATS product family. This will be made with the help of the implementation of the first part of the customization process presented in Appendix II. Thus, the solution for the customer inputs in our simulation will be generated using that program. The elicitation process and the solutions is shown in Part II of that appendix. However, below we will comment briefly the execution of that program to generate the solution. After that, the solution is used to transform the GPS into the custom ATS. Since our implementation does not cover this part of the customization process, it will be carried out manually.

Following the order in F' , the instantiation algorithm begins with the input d -functions, which by definition are enabled. As they are executed, the customer is asked about the electrical attributes of the utility and load, the timing of the power generator and its monitoring, the neutral grounding and the terminal block inclusion. Let us suppose, the customer have assigned the following values to those variables: $u = (110, 60, 1)$, $l = (40, 2)$, $g = (2, 0)$, $n = 1$ and $b = 0$. According to the instantiation pattern P_2 , these values completely define the customer requirements. Then, the next two d -functions in F' define the configuration of the transfer switch. $f_s(u, l)$ makes the assignment $s = (110, 50, (19, 85, 95))$, defining the electrical and dimensional characteristics of the transfer switch in reference to Table 2, and $f_p(n, l)$ makes the assignment $p = (1, 1, 0, 0)$, ascribing one pole for switching the neutral wires and one for the active wires, thus completing the configuration of transfer switch. Given that the customer did not require a terminal block, there is no need for its electrical specification and, therefore, the d -function $f_e(b, s)$ makes the assignment $e = \text{NA}$ (Not Applicable). The d -function $f_c(u, g)$ makes the assignment $c = (0, 0, 1, 0, 0, 1, 0, 0)$, transforming some of the specification related to the input variables u and g into a binary representation for the configuration of the 8-position DIP switch on the control board, 0 for “OFF” and 1 for “ON”. Since the level of fuel of the power generator will not be monitored, $f_a(c)$ makes the assignment $a = 0$, defining that there will be no buzzer in the customised ATS. Consequently, $f_h(a)$ makes the assignment $h = 0$, defining that the enclosure has no buzzer fixture. The next two d -functions complete the specification of the enclosure. $f_d(s, p)$ makes the assignment $d = (280, 480, 200)$, selecting from Table 3 the smallest enclosure for which all the other components can be assembled inside. $f_m(d)$ makes the assignment $m = \text{metal}$, defining the material of the enclosure (also in reference to Table 3) by selecting the first type of material related to the item that matches the dimensions just defined. Finally, the d -function $f_w(a, b, s, p)$ makes the assignment $w = ((P_1, P_2), (T_1, T_2), (M_1, SU_1), (M_N, SU_N))$, which corresponds to the wires (from the master template) required for the connections between the

components in the custom ATS. This d -function also specifies the cross-section area for the wires connected to the switch, based on a table relating the cross section and amperage of the wires (see Table I-2 in Appendix I).

The next stage in the instantiation process is the use of the solution just found to derive a member of the ATS product family from its GPS. As we have already noted, the instantiation of the GPS could have been carried out while the instantiation of the CN-F model progresses. The optional component types can be removed as soon as the inclusion variables have their values assigned, and the generic component types can be replaced by specific components as soon as all the variables on them have been assigned their values.

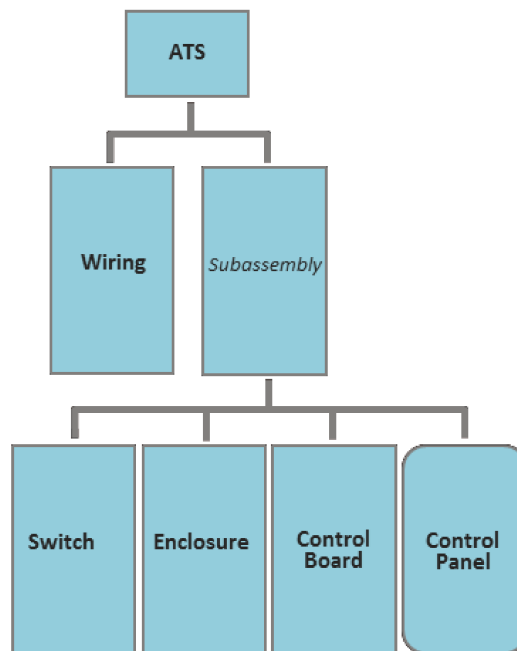


Figure 15. The architecture of the custom ATS

Based on the two-stage process, the instantiation of the GPS of the ATS product family begins with the removal of the optional leaf components Terminal Block and Buzzer. This is due to the fact that $b = 0$ and $a = 0$, respectively. Since removing these two components does not leave higher-order components in the hierarchy depleted, there is no further removal of components from the GPS. The resulting structure of the custom ATS is shown in Figure 15.

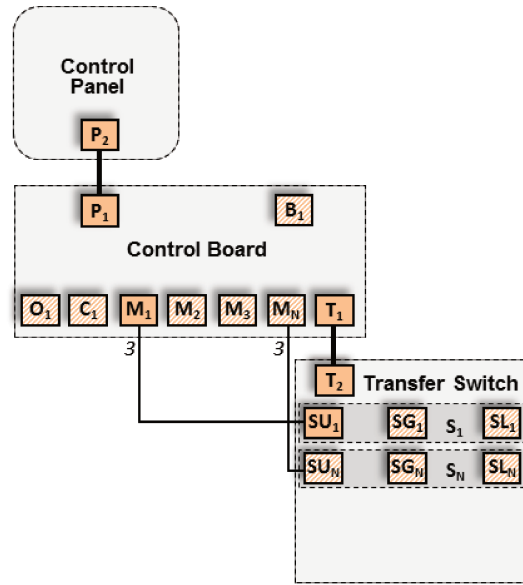


Figure 16. The wiring component for the custom ATS

To complete the transformation of this structure into a physical model representing the custom ATS, the generic leaf components need to be substituted by specific ones, defined by the values of remaining variables. Since the Control Panel is a specific/common component type, the same component is used for all members of the ATS product family. The Switch is specified as an 110V/50A switch with two poles. The specification of the Control Board is made by the configuration (OFF, ON, OFF, OFF, OFF, ON, OFF, OFF) of its 8-position DIP switch. The Enclosure is specified as a 280, 480 and 200 mm metal box with no hole on it, since the buzzer will not be required. Finally, the diagram of Figure 16 specifies the Wiring, thus completing the custom ATS.

Chapter 5

Related approaches to the customisation of product families

An early proposal for representing the customisation process of product families was the generic bill-of-material (GBOM) concept, introduced in [38; 39]. As a generalization of the bill-of-material structure used in production engineering, the GBOM avoids data redundancy by ascribing a common structure to the product family. It is a hierarchical structure composed of generic assemblies and generic primary products (or components), and the product family variability is represented by parameters. Members of the product family are derived from the GBOM with the help of a complementary decision tree and by a parameter inheritance mechanism to identify a specific variant, which satisfies the customer requirements. Moreover, using the GBOM concept both the commercial and assembly views on the product family are integrated into a single generic product model [40]. In [34] the product family architecture is analysed taking into account the functional, technological and physical perspectives alongside the development process of products. It is proposed that to represent a product family in a non-redundant way, a generic product structuring language should allow for the decomposition of product families into a hierarchy of generic compound and generic primitive products, which are ultimately decomposed into sets of primitive variants with identical interfaces. Moreover, this modelling language should allow for the coordination of dependent component variants in order to derive product family members.

Further contributions to the modelling of product families and their role for mass customisation are found in [4, 41, 42, 43, 44]. The approach is also centred on the architecture as “the logical organization of the product family from both the sales and engineering viewpoints.” Product family

variants are derived from a GPS as instantiations. This process begins within the sales perspective where optional functional features are selected to represent the customer requirements. At this stage, selection constraints are applied to prevent unfeasible options. Then, within the engineering perspective, these functional options are represented as variety parameters and propagated along the GPS by inheritance and derivation mechanisms. Basic variety generation methods such as attaching/removing, swapping and scaling modules are modelled by variable structural relationship and include module conditions. In [45, 46] the mechanism for instantiating the GPS is implemented as a graph rewriting system. Variants are derived from a base product by the operations of attaching, removing, swapping and scaling modules, carried out by production rules. However, in this approach, the control mechanism that specifies the applicable productions and their order of invocation to get all its correspondent variants is specific to the end-product and each compound component in the GPS. Moreover, except for the scaling operation, which admits a continuous range of transformations, all variants must be identified explicitly.

Another approach to the concept of product families based on graph grammar is presented in [47]. In this approach, the product family architecture is defined by a common core function structure and a set of optional functions. The members of the product family are derived by the application of grammar rules to add optional functions to the core function structure and to transform the resulting function structure into a product structure. Variety is obtained by the addition of different optional functions and by the use of alternate rules to transform functions into structure. Customer requirements are used in the selection of optional functions and to generate the final product structure/assembly viewpoint [48].

Product configuration is a research area in AI. It is defined as a design task in which components are selected from a set of pre-defined components and combined to meet a set of requirements, taking into account a set of design constraints that restricts their combination [15]. Approaches for representing and solving product configuration problems have been classified as rule-base, model-base and case-base paradigms [49]. Among the model-base approaches, those using constraints are akin to our approach and will be reviewed next.

For a product configuration problem applying the classical constraint satisfaction problem (CSP) the properties and ports of components are typically represented as variables, each one associated to a finite set of possible values, and subject to a set of design constraints, which restricts the values that can be assigned to the variables simultaneously. The design space is defined as the set of all possible combinations of values assigned to the variables. A valid configuration (or solution in the

design space) is an instantiation of the variables that do not violate any of the design constraints. Solutions are found by search procedures that typically combine backtracking and constraint propagation methods, using heuristics to improve their performance [35, 50]. However, as pointed out in [51], although classical CSP is a promising modelling approach, some extensions are necessary to cope with the specificities of product configuration problems. Hence, since the proposal made by Mittal and Frayman [52] to represent product configuration as a CSP problem, a number of other proposals to extend the CSP model have been put forward.

Because the set of variables that are relevant for the solution of a configuration problem may change dynamically during the problem solving, Mittal and Falkenhainer [53] proposed a dynamic constraint satisfaction problem (DCSP) approach. Besides the traditional constraints over the value of the variables, which they call Compatibility Constraints, in this approach it is introduced the notion of Activity Constraints. These constraints describe the conditions under which variables may or may not be activated to be part of the final solutions. They also propose an instantiation algorithm that can deal with both types of constraints.

To deal with the structural aspect of configuration problems, such as the constituent parts of the final product or the internal structure of components, in [53] it is proposed a composite CSP where variables are allowed to represent entire sub problems. According to Veron et al. [51], product configuration problems require the management of the product structural decomposition and the states of its components, the interaction of the user during the configuration process, and the mapping of functional requirements onto product components. To handle these requirements, they propose to model the configurable product as a tree with internal nodes representing sub-configurable components and leaf nodes corresponding to elementary configurable or standard components. The attributes of the configurable components are represented as variables and each component is associated to a state variable. These variables have four possible values: inactive, optional, required and completed. Constraints are used to restrict the combination of variables of the configurable components. A subset of the constraints is the state conditions, which handle the values of the state variables. The configuration process works on two levels. First, the state variables are used to manage the tree structure. Then, the CSP problem is addressed to define the attributes of the active components. The user expresses his choices by adding/retracting unary constraints.

CSP approaches have been focused mostly on discrete variables and binary constraints. However, in the configuration of engineering products it is quite common to have continuous variables and constraint on multiple variables. Thus, Gelle et al. [55] introduced local consistency methods to

handle discrete and numerical variables and in the same framework to address engineering products represented as a CSP.

In [56] it is proposed the Dependent CSP. In this approach, the variables can be related by dependencies or constraints and are divided into independent and dependent by means of the relation of dependency. The independent variables are assigned values from their associated domains, while the values of the dependent variables are assigned from the values of the independent variables through the relations of dependency. A solution is an assignment to the variables such that all dependencies and constraints are satisfied. The search for solutions is made by a backtracking method of the type "backjumping". The updating of values and the verification of constraints is organized by a directed acyclic graph. This graph is defined based on the dependencies between variables and of constraints in relation to the independent variables. Heuristics are used to establish the order in which variables are considered.

As we have noticed in Subsection 2.2.1, if the response time of a configurator to the inputs of the customers is too long, and the customer is required to undo previous decisions because the configuration process has reached a dead end, the interaction with the configurator will inevitably become a bad experience for the customer. These conditions arise if the configurator have to solve a computationally hard problem on real time whenever requirements are inputted, and the problem-solving process is not backtrack-free. To avoid these problems associated to search-based methods, some recent works resorted to a two-stage process, by precompiling all the solutions using some form of efficient representation. Although these methods still have to solve a hard problem to find all the solutions, this is done offline and only once. Then, the interactive part of the configuration process can be done efficiently. For instance, in [10] the solution space is encoded using binary decision diagrams. Although they claim to have very good practical results, depending on the size of the configuration problem they may run out of space, since their method compiles all the solutions of the problem. In [57], it is proposed a pre-processing method that result in a backtrack-free representation. Unlike other conventional approaches that add constraints to the problem, thus making them susceptible to space limitation, they remove values from the domain of the variables to make their representation of the problem backtrack-free. The disadvantage of this method is that solutions are lost.

In the next chapter, we will be considering the main advantages of our approach. However, it is worth at this point to make some considerations on how the proposal of this thesis compares to the approaches review above.

Although the concept of a GPS composed of component types can be found in the literature, the

one introduced in this work is a generalization and extension of the prevailing ones. In general, components are divided into common and variants. The former class is composed of the components that belong to all members of the product family and frequently are regarded as specific and fixed. The latter class is further divided into generic and optional components. However, in our classification scheme we have stressed the fact that common components can be generic, the generic/common component type. This implies that even the common part of the GPS, which forms the backbone of the product family, is also allowed to vary. Moreover, in contrast to the rather informal definitions of product families found in the literature, in our approach we provide a formal and strict definition that delimits the products that can belong to the product family, by requiring them to be isomorphic and coherent to a GPS.

To overcome the limitations of the CN model to deal with the specificities of product configuration problems, we introduced in this work the CN-F model as an extension of the classical CN model by the attachment of d -function to the variables. By means of the d -functions we have embedded procedural knowledge to an otherwise declarative framework. Besides the generation of the values for the variables during the instantiation process, d -functions are used to establish the dependencies between variables. Dependencies are typically revealed by means of tools such as Dependency Structure Matrixes (DSM) during the design process of product families [58, 37, 59]. However, through the d -function we describe the way the variables actually depend on each other.

Finally, differently from the other approaches that claim to be backtrack-free, we do not resort to computational power, nor do we require a pre-processing stage, but we take advantage of the knowledge about the design of the product families to systematise the customisation process. As a result, a configurator based on our approach can be implemented as dataflow programs, with the prospect of running even on handheld devices.

Chapter 6

Conclusions and Future Research

In this thesis, we have proposed a formal approach to the customisation of product families, thus providing a solid foundation for our claims. We have introduced a new knowledge framework for representing product families, which combines the GPS and the CN-F models. Deriving product family members from this framework is a two-stage process. First, a solution to the CN-F model is searched for, guided by instantiation patterns. Then, in the second stage, the solution is used to transform the product family GPS into a specific model. The outcome is a member of the product family that satisfies the customer requirements. We also defined conditions for which this process becomes a backtrack-free process.

Many of the requirements for dealing with product configuration which have been reviewed in the previous chapter, are present in our own approach. However, what is remarkable about it is that much of these requirements have been accomplished in a single framework. Moreover, we also have introduced many unique contributions, which have been related in Section 1.1. Among the advantages of our approach, it should be mention that:

- The approach is suited for the configuration of complex products for which the customers do not have the necessary expertise to participate directly in all the configuration process;
- It can deal with configuration problems for which the constraints between the variables are complex;
- It can deal with mixed discrete and continuous variables;

- It provides flexibility to the customisation process because the values of the variables need not to be predefined;
- Moreover, in principle, components may be designed during the customisation process (see the discussions below);
- Based on our approach, configurators can be implemented as dataflow programs requiring little computation power compared to other approaches that claim to be backtrack-free.

This approach has been developed to serve as a basis for the implementation of configurators to mass customization systems for manufacturable goods. However, it can equally well be applied to the area of software, where there has been much research in software product lines [60]. Moreover, because the configurator can be implemented as a dataflow program requiring little computation power, our approach might be very useful for other applications such as self-configuring systems and autonomous devices [61, 62, 63].

6.1 Applicability to the area of mass customisation

Concerning the applicability of our approach to practical problems, besides the ATS example presented in this work, it has been used in two other applications. In Schneider et al. [64] we made the configuration of a solar power pump system (SPPS). The task was to configure the SPPS so that it could pump water from a well to meet with the water requirements of the application, taking into account the required system autonomy and other conditions. Besides the choice of the water pumping system components, it is capable of defining the optimum arrangement of the photovoltaic and battery arrays and their wire interconnections. The CN-F model for this problem has a mix of discrete and continuous variables. The configurator was implemented as dataflow program, and ported to a simulator for a handheld device without compromising its performance. In Schneider et al. [65], we used our approach in connection to additive manufacturing technologies, or 3D printing, as they are popularly known. In that application, we have developed a customization system for a scale compressed air engine. After requirements such as the cylinder diameter and throw length have been entered into the system, it automatically dimensions all the related components of the CA engine accordingly. We have shown that our approach can be integrated naturally to a CAD system, resulting in a powerful product family representation system. Besides specifying the member of the product family that meets the customer requirements, the configurator also generates files containing the

detailed design of the product components. These files can be sent directly to the production system. As in the SPPS application described above, the CN-F model can also be modelled to satisfy the conditions for a backtrack-free customization process.

The capability of designing components during the customisation process, either automatically or by the customer, has a fundamental implication on the nature of the design problem our approach can deal with. Product configuration is defined as a design problem for which the components are specified in advance, either as an explicit or parameterised set of components [15]. On the other hand, for our approach, it is not necessary to know beforehand all the components that are associated to a generic component type. This follows from the fact that for a sufficient complex d -function, one may not know all the possible outcomes that can arise in practice. Therefore, we can conclude that our approach can be used to deal with design problems outside the domain of product configuration.

6.2 Implications for the routinization of the design activity

The concept of dependency pattern, introduced in this work, is the outcome of the dependencies between variables set up by the d -functions. The establishment of the instantiation patterns may be associated to the routinization of the problem-solving process that happens when one gets used to solve design problems of the same type [66, 14]. Although this issue was not our focus in this work, it is interesting to note that some of the results we get are corroborated by traits that are attributed to routine design. To exemplify this point, next we present a selection of some of those traits listed in Brown [14]. The material within brackets are the concepts in our approach that related to those traits.

- “Use of a fixed set of well-understood design plans [the instantiation patterns].”
- “Dependencies between sub problems [defined by the d -functions] are known (...).”
- “Sub problems can usually be solved in a fixed order [set by the instantiation patterns] with little or no back-tracking, due to the anticipated dependencies.”
- “The knowledge needed to calculate or select a value for each attribute [the d -functions] is known in advance.”

All those traits are clearly associated to our own findings. Actually, we have gone further setting conditions for which the problem-solving process becomes completely backtrack-free. Thus, we may conclude that our approach can be used as a theory to explain how routinization is established as the designer gets familiar with the design of a set of similar products that constitutes a product family.

6.3 Further developments and future research

Our main concern in this thesis was with the problem-solving core of the customization process. Therefore, some important issues were touched only slightly in this thesis, and must be further developed so that the practical potential of our approach can be fully exploited. These issues are related to the front and back end of the elicitation process. In the front end, one of the important issues is the customer interface. In our prototype configurator shown in Appendix II, we only implemented a rudimentary interface (below we discussed another issues regarding the interface). Another issue related to the front end of the elicitation process is the automatic generation of quotations [67, 68]. Although we have not considered this issue in any detail in this work, it should be noted that after the custom product is specified, all its components are known, and this is the starting point from which the quotation for the customer can be generated. Concerning the back end process, the generation of the product order is another important issue [67]. Once again, after the specification of the custom product is given, it can be further transformed into a suitable representation that can be used by the manufacturing system. In [65] we have proposed the integration of a product specification system, implemented using our approach, with a commercial CAD system for the generation of 3D models of the components of the custom compressed air engine. Subsequently, these files are saved as STL files, and send directly to the manufacturing system (based on 3D printers) for production. However, in general, the generation of production order will require a more elaborate representation of the GPS, and consequently, for the specific product models derived from it [69].

Finally, let us examine more closely co-design, one line of research that we intend to pursue in the future. Co-design has been much emphasized in the literature of mass customisation. It is defined as the integration of the customer “into value creation by defining, configuring, matching, or modifying an individual solution” [7]. However, instead of just presenting the customer with options to choose from, as it is typical for configurators, to exploit co-design in a broader sense, it is necessary to allow the customers to participate more deeply into the design stage of the mass customisation process, for example, by allowing them to directly designing parts of the product they need. In our approach, this can be done through the input d -functions, for example, by providing the customers with access to templates in a CAD system so that they can adapt components to their needs. Hints on this type of capability and its potential may be found in [70, 71].

Nevertheless, if too much flexibility is provided to the customer, the customization process may get in trouble at the design or manufacturing stage. Even if no apparent problem arises during those stages, there is no guarantee that the resulting product will have the desired performance or that it will

work at all. For example, it was part of our experience with the compressed air engine [65] that solutions that work well for a range of parameter values, may present functional problems if their limits are stretched too much. If we cannot be sure about the manufacturability, functionality and reliability of the custom product, it will be necessary to go through testing. In this case, the responsiveness of the mass customisation system will become seriously compromised. Therefore, the capability of defining new components or even extending the product family limits on the design space must be approached with caution. Actually, it is our belief that product customization apart from the more routine design processes tends to be a difficult and illusive task for a mass customization system, leaving little room for novel design [72], except for simple products.

Hence, with the increased flexibility given to the customer during the customisation process, it must be necessary to find ways to define the limits of the product family in the design space. However, this must be done so that the customization process does not become a frustrating tentative-and-error process to the customer. Moreover, the output of this customization process must be manufacturable functional, reliable, and it must perform as expected. In other words, it is necessary that the representation of the product family be certified by construction. This is a challenging issue that will extend the understanding of the capabilities of the customisation approach presented in this work, and one of our future lines of research.

References

- [1] K. Lancaster, “The Economics of Product Variety: A Survey,” *Marketing Science*, vol. 9, no. 3, pp. 189–206, Summer 1990.
- [2] M. V. Martin and K. Ishii, “Design for variety: a methodology for understanding the costs of product proliferation,” in *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering Conference*, August 18-22, 1996, Irvine, California
- [3] F. Salvador, C. Forza and M. Rungtusanatham, “Modularity, product variety, production volume, and component sourcing: theorizing beyond generic prescriptions,” *Journal of Operations Management*, vol. 20, pp. 549–575, 2002.
- [4] M. M. Tseng and J. Jiao, “Design for Mass Customisation,” *CIRP Annals*, vol. 45, no. 1, pp. 153–156, 1996.
- [5] J. Jiao, T. W. Simpson and Z. Siddique, “Product family design and platform-based product development: a state-of-the-art review,” *Journal of Intelligent Manufacturing*, vol. 18, no. 1, pp. 5–29, 2007.
- [6] B. J. Pine, *Mass Customization*, Boston: Harvard Business School Press, 1993.
- [7] F. T. Piller, “Mass Customisation: Reflections on the State of the Concept,” *International Journal of Flexible Manufacturing Systems*, vol. 16, no. 4, pp. 313–334, 2004.
- [8] R. Bourke, “Product Configurators: Key Enabler for Mass Customisation - An Overview,” *Midrange ERP*, August 2000.
- [9] N. Franke and F. T. Piller, “Key research issues in user interaction with user toolkits in a mass customisation system,” *The International Journal of Technology Management*, vol. 26, no. 5/6, pp. 578–599, 2003.
- [10] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, H. Hulgaard and J. Moller, “Fast backtrack-free product configuration using a precompiled solution space representation,” in *International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems*, Technical University of Denmark, Lyngby, Denmark, June 28–29, 2004.
- [11] T. W. Simpson, B. Aaron, L. A. Slingerland, S. Brennan, D. Logan and K. Reichard, “From user requirements to commonality specifications: an integrated approach to product family design,” *Research in Engineering Design*, vol. 23, no. 2, pp. 141–153, 2012.

- [12] T. W. Simpson, C. C. Seepersad and F. Mistree, "Balancing commonality and performance within the concurrent design of multiple products in a product family," *Concurrent Engineering: Research and Applications*, vol. 9, no 3, pp. 177–190, 2001.
- [13] K. Fujita, "Product Variety Optimization," in *Product Platform and Product Family Design Methods and Applications*, T. W. Simpson, Z. Siddique and J. R. Jiao, Eds. New York: Springer, 2006.
- [14] D. C. Brown, "Routineness Revisited," in *Mechanical Design: Theory and Methodology*, M. Waldron and K. Waldron, Eds. Heidelberg, Springer, 1996, pp. 195–208.
- [15] B. Wielinga and G. Schreiber, "Configuration design problem solving," *IEEE Expert*, vol. 12, no. 2, pp. 49–56, 1997.
- [16] T. W. Simpson, "Product platform design and customization: Status and promise," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 18, pp. 3–20, 2004.
- [17] Z. Pirmoradi, G. Wang, T. W. Simpson, "A Review of Recent Literature in Product Family Design and Platform-Based Product Development," in *Advances in Product Family and Product Platform Design: Methods & Applications*, T. W. Simpson, J. Jiao, Z. Siddique, K. Hölttä-Otto Eds. London: Springer, 2013.
- [18] S. Sanderson, and M. Uzumeri, "Managing product families: The case of the sony Walkman," *Research Policy*, vol. 24, no. 5, pp. 761–782, 1995.
- [19] D. Robertson and K. Ulrich, "Planning for Product Platforms," *Sloan Management Review*, vol. 39, no. 4, pp. 19–31, 1998.
- [20] M. H. Meyer and A. P. Lehnerd, *The Power of Product Platforms: Building Value and Cost Leadership*. New York: Free Press, 1997.
- [21] K. Ulrich, "The role of product architecture in the manufacturing firm," *Research Policy*, vol. 24, no. 3, pp. 419–440, 1995.
- [22] J. K. Gershenson, G. J. Prasad and Y. Zhang, "Product modularity: definitions and benefits," *Journal of Engineering Design*, vol. 14, no. 3, pp. 295–313, 2003.
- [23] G. Pahl and W. Beitz, *Engineering Design: A Systematic Approach*. London: Design Council, 1988.
- [24] K. Ulrich and K. Tung, "Fundamentals of product modularity," in *ASME Winter Annual Meeting Symposium on issues in Design/Manufacturing Integration*, Atlanta, GA, November 1991.
- [25] K. Fujita, H. Sakaguchi and S. Akagi, "Product variety deployment and its optimization under modular architecture and module communalization," in *Proceedings of the ASME design*

- engineering technical conferences* (DETC99/DFM-8923), 12–15 September 1999, Las Vegas, Nevada. New York: ASME, 1999.
- [26] P. Zipkin, “The Limits of Mass Customization,” *Sloan Management Review*, vol. 42, no. 3, pp. 81–87, Spring 2001.
 - [27] G. da Silveira, D. Borenstein and F. S. Fogliatto, “Mass customisation: literature review and research directions,” *International Journal of Production Economics*, vol. 72, no. 1, pp. 1–13, 2001.
 - [28] F. S. Fogliatto, G. J. C. da Silveira and D. Borenstein, “The mass customisation decade: An updated review of the literature,” *International Journal of Production Economics*, vol. 138, pp. 14–25, 2012.
 - [29] F. T. Piller, P. Schubert, M. Koch and K. Möslin, “Overcoming Mass Confusion: Collaborative Customer Co-Design in Online Communities,” *Journal of Computer-Mediated Communication*, vol. 10, no. 4, July 2005.
 - [30] H.M. Khalid and M. G. Helander, “Facilitating Mass Customization and Web-based Do-It-Yourself Product Design,” in *The Customer Centric Enterprise: Advances in Mass Customization and Personalization*, M.M. Tseng and F.T. Piller, Eds. Berlin: Springer, 2003.
 - [31] F. T. Piller and M. M. Tseng, “New directions for mass customization,” in *The Customer Centric Enterprise: Advances in Mass Customization and Personalization*, M.M. Tseng and F.T. Piller, Eds. Berlin: Springer, 2003.
 - [32] R. Duray and G. W. Milligan, “Improving customer satisfaction through mass customisation. *Quality Progress*, vol. 32, no. 8, pp. 60–66, 1999.
 - [33] A. Haug, K. Ladeby and K. Edwards, “From engineer-to-order to mass customisation,” *Management Research News*, vol. 32, no. 7, pp. 633–644, 2009.
 - [34] F. Erens and K. Verhulst, “Architectures for product families,” *Computers in Industry*, vol. 33, no. 2–3, pp.165–178, 1997.
 - [35] R. Dechter, “Constraint Networks,” in *Encyclopedia of Artificial Intelligence*, S. C. Shapiro, Ed. New York, Wiley, pp. 276–285, 1992.
 - [36] D. V. Steward, *System Analysis and Management: Structure, Strategy and Design*, New York: Petrocelli Books, 1981.
 - [37] D. M. Sharman and A. A. Yassine, “Characterizing complex product architectures,” *Systems Engineering*, vol. 7, no. 1, pp. 35–60, 2004.

- [38] E. A. van Veen, *Modelling Product Structures by Generic Bills-of-Material*, PhD thesis, Eindhoven University of Technology, 1991.
- [39] H. M. H. Hegge and J. C. Wortmann, "Generic Bill-of-Material: A New Product Model," *International Journal Production Economics*, vol. 23, pp.117–128, 1991.
- [40] H. C. Wortmann and F. J. Erens, "Control of Variety by Generic Product Modeling," in *Proceedings of The First World Congress on Intelligent Manufacturing Processes and Systems*, University of Puerto Rico, 13–17 February, 1995, vol. 2, pp. 1327–1342.
- [41] M. M. Tseng and J. Jiao, "Design for mass customisation by developing product family architecture," in *Proceedings of the ASME design engineering technical conference (DETC98/DTM-5717)*, 13–16 September 1998, Atlanta, Georgia. New York: ASME.
- [42] M. M. Tseng and J. Jiao, "Fundamental Issues Regarding Developing Product Family Architecture for Mass Customisation," *Integrated Manufacturing Systems*, vol. 11, no. 7, pp. 469–483, 2000.
- [43] J. Jiao and M. M. Tseng, "Fundamentals of product family architecture," *Integrated Manufacturing Systems*, vol. 11 no.7, pp. 469–483, 2000.
- [44] X. Du, J. Jiao and M. M. Tseng, "Architecture of product family: Fundamentals and methodology," *Concurrent Engineering: Research & Applications*, vol. 9, no. 4, pp. 309–325, 2001.
- [45] X. Du, M. M. Tseng and J. Jiao, "Graph grammar based product family modeling," *Concurrent Engineering Research & Applications*, vol. 10, no. 2, pp.113–128, 2002.
- [46] X. Du, M. M. Tseng and J. Jiao, "Modelling platform-based product configuration using programmed attributed graph grammars," *Journal of Engineering Design*, vol. 14, no. 2, pp. 145–167, June 2003.
- [47] Z. Siddique and D. W. Rosen, "Product platform design: A graph grammar approach," in *Proceeding of the 11th International Conference in Design Theory and Methodology*, Las Vegas, NV, Paper No. DETC99/DTM-8762, 1999.
- [48] Z. Siddique and K. R. Boddu, "A mass customization information framework for integration of customer in the configuration/design of a customized product," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 18, pp. 71–85, 2004.
- [49] D. Sabin and R. Weigel, "Product Configuration Frameworks – A Survey," *IEEE Intelligent Systems*, vol. 13, no. 4, pp. 42–49, 1998.

- [50] I. Miguel and Q. Shen, "Solution Techniques for Constraint Satisfaction Problems: Foundations," *Artificial Intelligence Review*, vol. 15, no. 4, pp. 243–267, 2001.
- [51] M. Veron, H. Fargier and M. Aldanondo, "From CSP to Configuration Problems," in *AAAI-99 Workshop on Configuration*, Orlando, Florida, July 18–19, 1999.
- [52] S. Mittal and F. Frayman, "Towards a Generic Model of Configuration Tasks," in *Proceedings of the 11th International Joint Conference of Artificial Intelligence*, San Francisco: Morgan Kaufman, 1989, pp.1395–1401.
- [53] Mittal, S. and Falkenhainer, B., "Dynamic Constraint Satisfaction Problems," in *Proceedings of the 8th National Conference on Artificial Intelligence*, 1990, pp. 25-32.
- [54] D. Sabin and F. Freuder, "Configuration as Composite Constraint Satisfaction," in *Technical Report FS-96-03, Workshop on Configuration*, Menlo Park: AAAI Press, 1996, pp. 28–36.
- [55] E. Gelle, B. V. Faltings, D. E. Clement, and I. F. C. Smith, "Constraint Satisfaction Methods for Applications in Engineering," *Engineering with Computers*, vol. 16, no. 2, pp. 81–85, 2000.
- [56] H. Xie, P. Henderson, J. Neelankavil and J. Li, "A Systematic search strategy for product Configuration," in *17th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems Manufacturing (IEA)*, Ottawa, Ontario, January 1, 2004.
- [57] E. C. Freuder, T. Carchrae and J. C. Beck, "Satisfaction Guaranteed," in *Workshop on Configuration, Eighteenth International Joint Conference on Artificial Intelligence*, 2003.
- [58] T. R. Browning, "Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions," *IEEE Transactions on Engineering Management*, vol. 48, no. 3, August 2001.
- [59] D. B. Luh, Y. T; Ko and C. H. Ma, "A structural matrix-based modelling for designing product variety," *Journal of Engineering Design*, vol. 22, no.1, pp. 1–29, 2011.
- [60] K. Pohl, G. Böckle, F. J. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Berlin, Springer-Verlag, 2005.
- [61] B. C. Williams, P. P. Nayak and U. Nayak, "A Model-based Approach to Reactive Self-Configuring Systems," In *Proceedings of AAAI-96*, 1996, pp. 971-978.
- [62] L. Feng, D. J. Chen and M. Torngren, "Self configuration of dependent tasks for dynamically reconfigurable automotive embedded system," in *47th IEEE Conference on Decision and Control*, 2008. CDC 2008, Cancun, 9-11 December 2008, pp. 3737–3742.
- [63] M. Zeller, C. Prehofer, G. Weiss, D. Eilers and R. Knorr, "Towards Self-Adaptation in Real-Time, Networked Systems: Efficient Solving of System Constraints for Automotive Embedded

- Systems", in *Fifth IEEE International Conference on Self-Adaptive and Self-Organizing Systems* (SASO), Ann Arbor, MI, October 2011, pp. 79 – 88.
- [64] H.M. Schneider, M.F. Espindola, Y. Iano, "How to Squeeze a Configurator into a Handheld Device," in *Proceedings of the 7th World Conference on Mass Customization, Personalization, and Co-Creation* (MCPC 2014), T.D. Brunoe et al. Eds. Aalborg, Denmark, February 4th - 7th, 2014, pp. 253-265.
- [65] H.M. Schneider, D.T. Kemmoku, P.Y. Moritomi, J.V. L. da Silva, Y. Iano, "Matching the Capabilities of Additive Technologies with a Flexible and Backtrack-free Product Family Customisation Process," in *Proceedings Fraunhofer Direct Digital Manufacturing Conference 2014*, Berlin, Germany, March 12-13, 2014.
- [66] J. S. Gero, "Design Prototypes: A Knowledge Representation Schema for Design," *AI Magazine*, Special Issue on Design, vol. 11, no. 4, pp. 26–36, Winter 1990.
- [67] C. Forza and F. Salvador, "Managing for variety in the order acquisition and fulfillment process: The contribution of product configuration systems," *International Journal of Production Economics*, vol. 76, pp. 87–98, 2002.
- [68] L. Hvam, S. Pape and M. K. Nielsen, "Improving the quotation process with product configuration," *Computers in Industry*, vol. 57, pp. 607–621, 2006.
- [69] A. Haug, L. Hvam and N. H. Mortensen, "A layout technique for class diagrams to be used in product configuration projects," *Computers in Industry*, vol. 61, pp. 409–418, 2010.
- [70] Y. Ariadi, R. I. Campbell, M.A. Evans and I.G. Graham, "Combining Additive Manufacturing with Computer-aided Consumer Design," in *23rd Solid Freeform Fabrication Symposium 2012*, D. L. Bourell, et al., Eds. Austin, Texas, 2012, pp. 238–249.
- [71] K. Hildebrand and M. Alexa, "Sketch-based pipeline for mass customization," in *ACM SIGGRAPH 2013 Talks*, Anaheim, California, July 21-25, 2013.
- [72] J. S. Gero and R. Sosa, "Complexity measures as a basis for mass customization of novel designs," *Environment and Planning B: Planning and Design*, vol. 35, no. 1, pp. 3–15, 2008.

APPENDICES

Appendix I

Modelling the Automatic Transfer Switch

In this appendix, we present the complete and detailed description of our knowledge framework for the automatic transfer switch (ATS) product family. This modelling is based on a commercial product developed at the Centre of Information Technology Renato Archer. Although some of its features were designed to allow reuse within a range of different powers, the goal of that project was not the development of an ATS product family as defined in this work. Hence, the ATS product family that will be presented here is a generalization of that product. Moreover, we modified some of its features to illustrate specific points of our theory of the customization process of product families.

In the following sections, first we give a description of the ATS product family. In Section I.2, we present the modelling of its generic product structure (GPS). In Section I.3, the elements of the constraint network model extended with *d*-functions (CN-F) for the ATS product family are detailed. After the specification of each of the *d*-functions we add some comments with respect to the consistency conditions, defined in Section 4.1.2 of the main body of the thesis. In Section I.4, we define the instantiation patterns and discuss their compliance to the consistency conditions. Finally, in Section I.5, we redefine the instantiation algorithm for CN-F models that satisfy both consistency conditions. Some parts of the material of this appendix have already been presented in the main body of the thesis to illustrate the concepts introduced there.

In Appendix II, we will present the implementation of a prototype of the configurator for the ATS product family based on the modelling presented here.

I.1 The ATS product family

The ATS is a device that senses the loss of power on the utility and promptly activates an emergency generator set to restore power to a vital load, such as emergency lights, security equipment, etc. The load will be automatically transferred back to utility when its power has been restored. Besides these main functions, the ATS may perform some auxiliary functions. For example, it may run the engine

periodically, as a maintenance procedure, and monitor the fuel level of the engine, sounding an alarm if the level is low. The ATS product family of our example is focused at residential and small business applications.

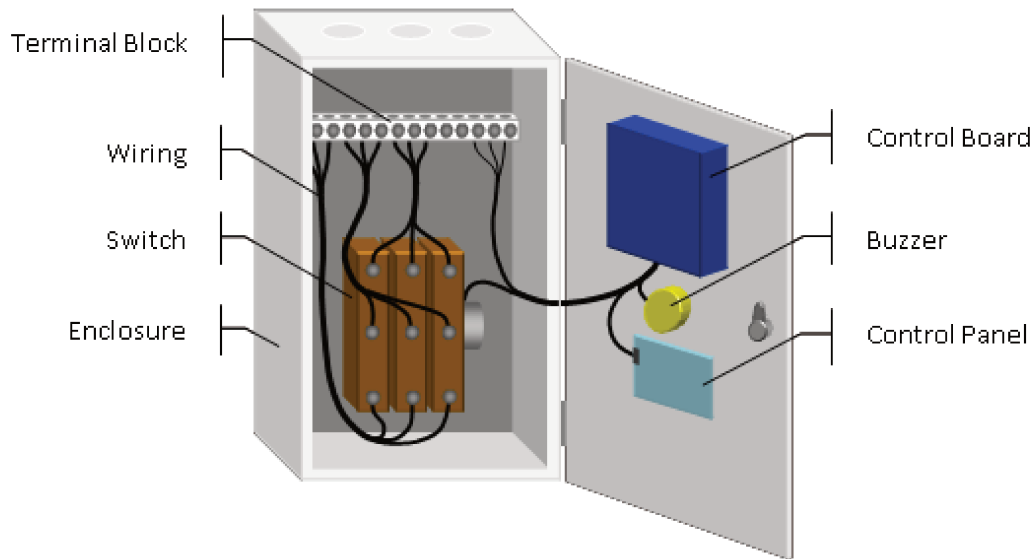


Figure I-1. The ATS and its main components

As depicted in Figure I-1, the ATS is composed from seven main components: transfer switches, control board, control panel, buzzer, terminal block, wiring and enclosure. The transfer switch is composed of up to four poles, one for every wire of the load that must be switched between the utility and the alternative energy source. The control board is the electronic circuit that is responsible for the realization of the ATS functions. Its main functions are sensing the utility, starting/stopping the engine-generator group, commuting the transfer switch and the maintenance of the power generator. The control panel contains the buttons and light indicators for the basic operation of ATS. The buzzer is the device for sounding alarms in case of operation failures. The role of the terminal block is to provide an interface to facilitate the electrical connection of the ATS to the utility, engine-generator group and load. Wiring is the bundle of wires that make all the electrical connection between the terminals inside the ATS. The role of the enclosure is to provide protective and structural support for the components of the ATS.

I.2 The generic product structure

According to our formal definition, presented in Section 3.2 of the main body of the thesis, the GPS is a

generalization of the physical models for all the members of the product family. It stands as the product structure for the product family. Instead of physical components, the GPS is composed of classes of components or component types. In our approach, we used a simple diagram as the representation for the GPS. It provides the general pattern for the assembling of the components of the product family members. The components of the GPS are classified into four component types: generic/common, generic/optional, specific/common and specific/optional. These categories identify the mode of variation of the product family.

Figure I-2 shows the GPS (in the background) and its component types for the ATS product family. Except for the control panel, which is a specific/common component type, all other component types are customizable to meet the requirements of the customers. The transfer switch is a generic/common component type. It is rated with amperage equal or greater than the amperage of the load, and configured to have a pole for every wire of the load that must be switched. The control board is the logical unit of ATS. In our example, its functions are: monitoring of the utility, starting/stopping the engine, switching the load between utility and generator (taking into account the waiting time for engine warm) and monitor the battery charge and level fuel generator. It has an 8-position DIP switch that is configured to represent the voltage, frequency and the number of phases of the utility, as well the delay time before the load can be transferred to the power generator and whether the customer wants the power generator to be monitored. Thus, the control board is also a generic/common component type. The buzzer is a specific/optional component type. It is included in the ATS configuration if the monitoring of the power generator is one of the functions of the control board. The enclosure is a generic/common component type. Its dimensions will change depending on the dimensions of the components that go inside. The customer can also express his/her preferences by choosing the material of the enclosure. The terminal block is a generic/optional component type. The customer decides whether it will be included or not. If the terminal block is included, it is rated with the same amperage of the transfer switch. Finally, the wiring is a generic/common component type. The bundle of wires is configured to provide all the necessary electrical connections within the ATS, and the cross-section areas of the wires are rated to support the currents that will pass through them.

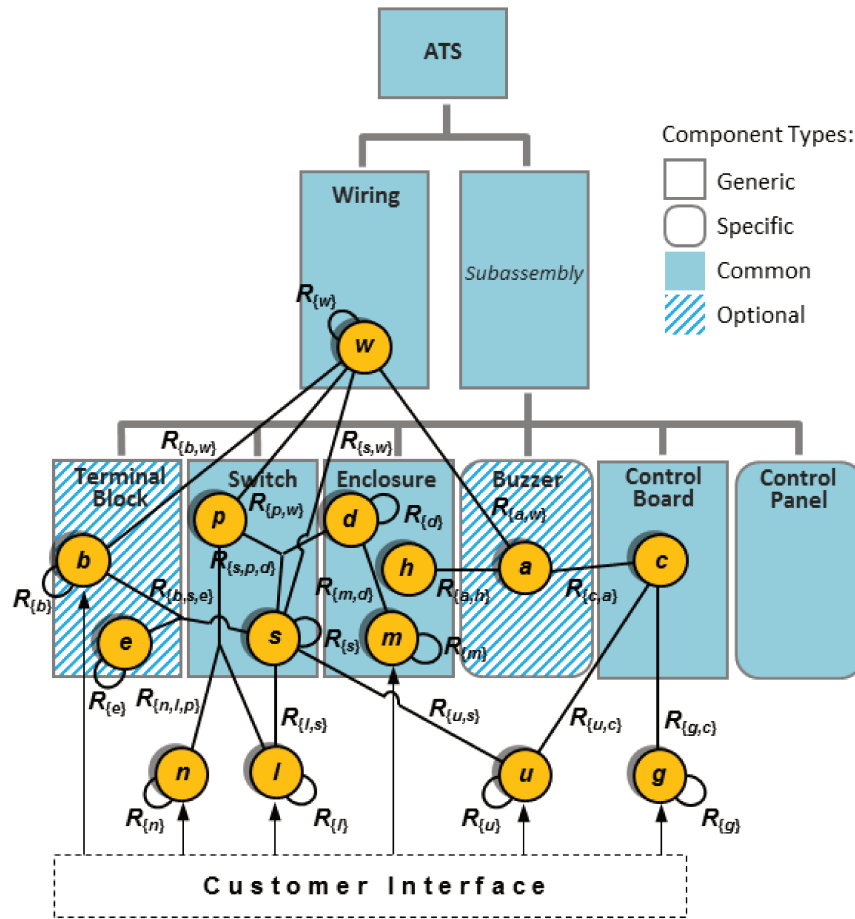


Figure I-2. The GPS (background) and the CN-F model (foreground) for the ATS product family

I.3 The constraint network model with extended functions

In the following three subsections, we will cover each of the elements comprising the CN-F model for the ATS product family.

I.3.1 Variables

For each variable of the CN-F model, we will describe its nature, composition and domain. For convenience, variables may be group to form composite variables. As depicted in Figure I-2, variables are represented as nodes of the constraint network. According to our approach, except for input variables, there is no need to define the domains explicitly. However, because of their simplicity, we have defined explicitly the domain of almost all the variables.

Utility (u)

The utility is the primary source of energy. It is characterized by a voltage (u_v), frequency (u_f) and number of phases (u_p).

$$D_u = \{(u_v, u_f, u_p): u_v \in \{110, 220\}, u_f \in \{50, 60\}, u_p \in \{1, 3\}\}$$

Engine-generator group (g)

The engine-generator group is the alternative source of energy. It is assumed that the engine-generator group is pre-sized to meet the electrical load, and all its output wires required by the ATS are available. Depending on the type of engine, such as gas, gasoline or diesel, there is a specific waiting time (g_w) to warm up the engine before the load can be transferred to the generator. At the customer discretion, the operational monitoring (g_m) of the engine-generator group can be done automatically ($g_m = 1$) or not ($g_m = 0$).

$$D_g = \{(g_w, g_m): g_w \in [0,15] \wedge g_m \in \{0,1\}\}$$

Emergency load (l)

The emergency load is the electric circuit which will be powered by the alternative energy source. It is characterized by the load current (l_a) and the number of wires (l_w) to be connected to the ATS. The voltage and number of phases of the load are assumed to be compatible with the utility.

$$D_l = \{(l_a, l_w): l_a \in (0,80], l_w \in \{2, 3, 4\}\}$$

Neutral grounding (n)

This variable refers to the grounding condition of the neutral from the utility and the generator. If both are grounded $n = 0$, otherwise, $n = 1$.

$$D_n = \{n: n \in \{0,1\}\}$$

Enclosure material (m)

This variable refers to the type of material of the ATS enclosure. Table I-1 introduces the available enclosures relating their dimensions and materials.

$$D_m = \{m: (_, m) \in \text{Table I-1}\}$$

Table I-1. Specifications of the available ATS enclosures

Item	Dimensions*	Material
1	(260, 190, 450)	metal
2	(260, 190, 450)	plastic
3	(280, 200, 480)	metal
4	(280, 200, 480)	plastic
5	(310, 210, 530)	metal
6	(310, 210, 530)	plastic
7	(350, 220, 600)	metal

*(w, h, l) in mm

Enclosure dimensions (*d*)

The role of the enclosure is to provide structural support and protection to the other components of the ATS. In this example, the enclosure will be available in a few standard dimensions, specified in Table I-1.

$$D_d = \{(w, h, l): ((w, h, l), _) \in \text{Table I-1}\}$$

Terminal block inclusion (*b*)

This variable accounts for the inclusion ($b = 1$) of the terminal block in the configuration of the ATS or not ($b = 0$).

$$D_b = \{b: b \in \{0,1\}\}$$

Terminal block amperage (*e*)

This variable specifies the nominal amperage of the terminal block, according to Table I-2. However, if the terminal block will not be included in the configuration of the ATS, $e = \text{NA}$.

Table I-2. Specifications of the available terminal blocks

Item	Amperage
1	30
2	50
3	80

$$D_e = \{e: e = \text{NA} \vee e \in \text{Table I-2}\}$$

Buzzer inclusion (a)

This variable accounts for the inclusion of the component Buzzer ($a = 1$) in the configuration of the ATS or not ($a = 0$).

$$D_a = \{a: a \in \{0, 1\}\}$$

Enclosure hole (h)

A hole must be made on the door of the enclosure ($h = 1$) to attach the buzzer. Otherwise $h = 0$.

$$D_h = \{h: h \in \{0, 1\}\}$$

Control board configuration (c)

The control board is configured through an 8-position DIP switch: $DS1$ for the monitoring of the engine, $DS2$ for the voltage of the utility, $DS3$ for its frequency, $DS4$ for the number of phases to be monitored, and $DS5, DS6, DS7, DS8$ to encode the waiting time. Since the waiting time can vary from 0 to 15 seconds, it has a four binary representation: (0, 0, 0, 0) stands for 0, (0, 0, 0, 1) for 1, (0, 0, 1, 0) for 2, and so on. Hence, values for the variable c are binary vectors corresponding to the possible configurations of the DIP switch, in the position OFF for 0 or ON for 1.

$$D_c = \{(DS1, DS2, DS3, DS4, DS5, DS6, DS7, DS8): DS_i \in \{0, 1\}\}$$

Switch poles (p)

The switch component is composed of poles, one for each wire of the load that must be switched between the utility and the generator. The switch may have one to four poles $P0, P1, P2$, and $P3$. $P0$ is associated to the neutral, and $P1, P2$ and $P3$ are associated to each of the possible phases of the utility.

If the neutral are not grounded, it is not required to be switched and then the pole $P0$ (for the neutral) is replaced with a solid bar. As a general rule, if switch represented by Pi is present in the ATS, then $Pi = 1$, otherwise, $Pi = 0$. Thus, the number of poles is equal to $\sum_{i=0}^3 Pi$.

The overall size of the component switch is a function of the number of poles and their dimensions. In the calculation of its total dimensions, the dimensions of the solid bar will be assumed to be negligible.

$$D_p = \{(S_0, S_1, S_2, S_3): S_i \in \{0, 1\}\}$$

Table I-3. Specifications of the available switches

Item	Amperage	Voltage	Dimensions*
1	25	110	(18,80,90)
2	25	220	(18,80,90)
3	50	110	(19,85,95)
4	50	220	(19,85,95)
5	80	110	(21,95,110)
6	80	220	(21,95,110)

*(w, h, l) in mm

Switch specification (s)

Each pole of the switch will be characterized by its nominal amperage (s_a), voltage (s_v) and physical dimensions (w, h, l). Table I-3 introduces the available switches.

$$D_s = \{(s_a, s_v, (w, h, l)): (s_a, s_v, (w, h, l)) \in \text{Table I-3}\}$$

Wiring configuration (w)

The wiring makes all the necessary electrical connections between the component terminals in the ATS. Figure I-3 shows the diagram that represents all those possible connections. Thick lines in the represent multiple wires, lines connecting solid terminal represent wires common to all family members and lines connecting hatched terminal are optional wires. If every wire in the diagram is represented by the pairs of terminals that it connects, the wiring diagram can be represented by the list

$WIRE_TEMPLATE = \{(P_1, P_2), (B_1, B_2), (O_1, O_2), (C_1, C_2), (SU_1, M_1), (SU_2, M_2), (SU_3, M_3), (SU_N, M_N), (SU_1, U_1), (SU_2, U_2), (SU_3, U_3), (SU_N, U_N), (SG_1, G_1), (SG_2, G_2), (SG_3, G_3), (SG_N, G_N), (SL_1, L_1), (SL_2, L_2), (SL_3, L_3), (SL_N, L_N)\}$. In this case, the wiring for a custom ATS is a subset of $WIRE_TEMPLATE$ by the removal of the pairs that correspond to unused wires. In our example, only the wires connected to the switch will have their cross-section area configured due to changes in amperage. This will be made according to Table I-4, which shows the diameter of the cross section area of the wire and the related amperage. The lengths of the wires will be regarded as fixed.

$$D_w = \{(WIRING, A): WIRING \subseteq WIRE_TEMPLATE \wedge (A, _) \in \text{Table I-4}\}$$

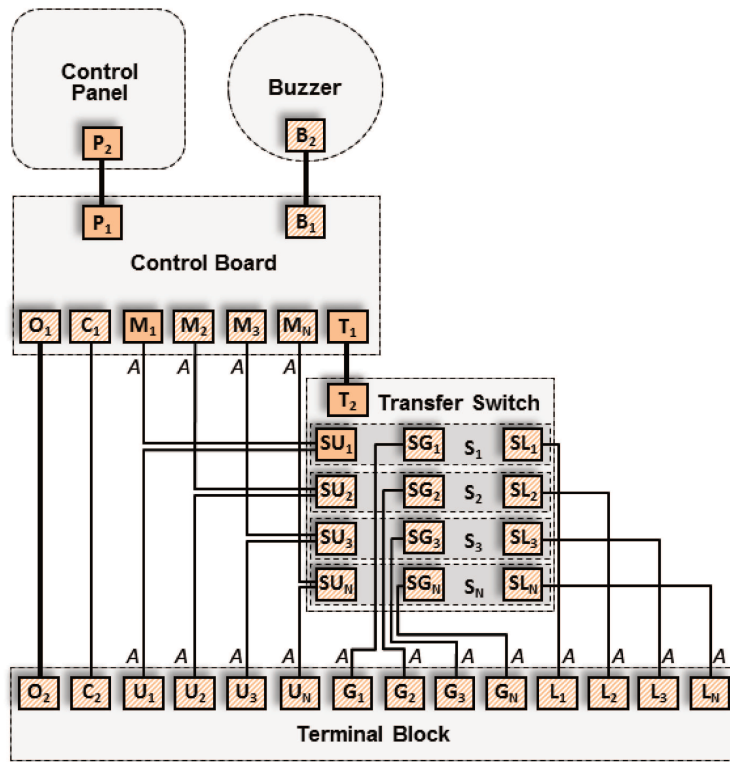


Figure I-3. Wire template for the ATS product family

Table I-4. Specifications of the cross section area of the available wires

Item	Diameter*	Amperage
1	2	30
2	3	50
3	4	80

*in mm

I.3.2 Constraints

Constraints define how the variables are related. The relations may be described by mathematical or logical expressions or explicitly, by a list of all the allowed combinations of values that can be assigned to the variables. The set of variables in subscript indicate all the variables that are being related by the constraint. The symbol \Leftrightarrow stands for “if and only if”.

$$R_{\{u\}} : u_v \in \{110, 220\}; u_f \in \{50, 60\}; u_p \in \{1, 3\}$$

$$R_{\{g\}} : 0 \leq g_w \leq 15; g_m \in \{0,1\}$$

$$R_{\{l\}} : 1 \leq l_a \leq 80, l_w \in \{2, 3, 4\}$$

$$R_{\{n\}} : n \in \{0,1\}$$

$$R_{\{m\}} : m \in \{\text{metal, plastic}\}$$

$$R_{\{b\}} : b \in \{0,1\}$$

$$R_{\{d\}} : ((w, h, l), _) \in \text{Table I-1}$$

$$R_{\{e\}} : e = \text{NA} \vee e \in \text{Table I-2}$$

$$R_{\{s\}} : (s_a, s_v, s_d) \in \text{Table I-3}$$

$$R_{\{w\}} : (A, _) \in \text{Table I-4}$$

$$R_{\{u,c\}} : u_v = 110 \Leftrightarrow DS2 = 0; u_v = 220 \Leftrightarrow DS2 = 1;$$

$$u_f = 50 \Leftrightarrow DS3 = 0; u_f = 60 \Leftrightarrow DS3 = 1;$$

$$u_p = 1 \Leftrightarrow DS4 = 0; u_p = 3 \Leftrightarrow DS4 = 1$$

$$R_{\{g,c\}} : g_m = 0 \Leftrightarrow DS1 = 0; g_m = 1 \Leftrightarrow DS1 = 1;$$

$$(g_w)_{10} = DS5 \times 2^3 + DS6 \times 2^2 + DS7 \times 2^1 + DS8 \times 2^0$$

$$R_{\{l,s\}} : s_a \geq l_a$$

$$R_{\{u,s\}} : s_v = u_v$$

$$R_{\{m,d\}} : (d, m) \in \text{Table I-1}$$

$$R_{\{a,h\}} : a = 1 \Leftrightarrow h = 1$$

$$R_{\{c,a\}} : DS1 = 1 \Leftrightarrow a = 1$$

$$R_{\{s,w\}} : A \geq s_a$$

$$R_{\{p,w\}} : S_2 = 0 \Leftrightarrow (SU_2, U_2), (SU_2, M_2), (SG_2, G_2), (SL_2, L_2) \notin WIRING;$$

$$S_3 = 0 \Leftrightarrow (SU_3, U_3), (SU_3, M_3), (SG_3, G_3), (SL_3, L_3) \notin WIRING$$

$$S_N = 0 \Leftrightarrow (SU_N, U_N), (SU_N, M_N), (SG_N, G_N), (SL_N, L_N) \notin WIRING$$

$$R_{\{a,w\}} : a = 0 \Leftrightarrow (B_1, B_2) \notin WIRING$$

$$R_{\{b,w\}} : b = 0 \Leftrightarrow (O_1, O_2), (C_1, C_2), (SU_1, U_1), (SG_1, G_1), (SL_1, L_1), (SU_2, U_2),$$

$$(SL_2, G_2), (SL_2, L_2), (SU_3, U_3), (SG_3, G_3), (SL_3, L_3), (SU_n, U_n), (SG_N, G_N),$$

$$(SL_N, L_N) \notin WIRING$$

$$R_{\{b,s,e\}} : b = 1 \Leftrightarrow e \geq s_a; b = 0 \Leftrightarrow e = \emptyset$$

$$R_{\{n,l,p\}} : S_N + S_1 + S_2 + S_3 = l_w - n$$

$$R_{\{s,p,d\}} : d_\alpha \geq \alpha; d_\beta \geq \beta; d_\gamma \geq \gamma; d_\delta \geq \delta; d_\varepsilon \geq \varepsilon$$

Comment: The size of the enclosure needs to be large enough to avoid conflicts between components and to facilitate their assembly. The determining factor for the size of the enclosure is the component switch, which is a function of the dimensions of the poles and the number. The switch is fixed at the back of the enclosure at distances $d_\alpha, d_\beta, d_\gamma, d_\delta$ and d_ε from the internal sides of the enclosure, as shown in Figure I-4. These distances should not be lower than a minimum, designated by $\alpha, \beta, \gamma, \delta$ and ε (in millimeters), respectively. These conditions are expressed by $R_{\{s,p,d\}}$. In our example, we will assume these limits to be 120, 120, 180, 180 and 110, respectively.

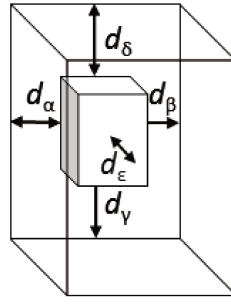


Figure I-4. Space allowances between the switch and the sides of the enclosure

1.3.3 Design functions

In what follows, we present the specification of the d -functions attached to the variables of the CN-F. They are defined as procedures and described in structured language. After each d -functions we add comments on their domain of definition and the incorporation of constraints in view of our considerations about the consistency conditions in Section I-4.

In the description of some d -functions, particularly those attached to the input variables, we make use of the procedure **Ask** (*Question*, *Options*, *Variable*), which ask the customer the question in *Question* to provide a value within the range defined in *Options*, and assigns the value chosen by the customer to the variable in *Variable*. In our example, *Options* is defined explicitly, by a set of alternatives, or implicitly, by the limits of an interval. Since the d -functions attached to the input variables do not depend on other variables, they will be represented without arguments.

f_u

Begin

1. **Ask** (What is the voltage of the utility?, {110, 220}, u_v)
2. **Ask** (What is the frequency of the utility?, {50, 60}, u_f)
3. **Ask** (What is the number of phases of the utility?, {1, 3}, u_p)
4. $u = (u_v, u_f, u_p)$

End

Comment: By this d -function the customer assigns the values for the voltage, frequency and the number of phases of the utility. The range of options presented to the customer corresponds to the values defined by the constraint $R_{\{u\}}$ and, therefore, this constraint is satisfied by the values generated by the d -function. Moreover, it is defined for all combinations of the options.

f_g

Begin

1. **Ask** (What is the waiting time of the generator?, $[0,15], g_w$)
2. **Ask** (Is the generator to be automatically monitored?, $\{yes, no\}, reply$)
3. If $reply = yes$, then $g_m = 1$, else $g_m = 0$
4. $g = (g_w, g_m)$

End

Comments: By this d -function the customer assigns the values for the waiting time and expresses his preference regarding the monitoring of the engine-generator group. The range of options presented to the customer corresponds (after the conversion) to the values defined by the constraint $R_{\{g\}}$. Moreover, this d -function is defined for all combinations of the options.

f_l

Begin

1. **Ask** (What is the amperage of the load?, $(0, 80], l_a$)
2. **Ask** (How many wires does the load circuit have? $\{2, 3, 4\}, l_w$)
3. $l = (l_a, l_w)$

End

Comments: By this d -function the customer assigns the values for the amperage and number of wires of the load. The range of options presented to the customer corresponds to the values defined by constraint $R_{\{l\}}$. Moreover, this d -function is defined for all combinations of the options.

f_n

Begin

1. **Ask** (Are the utility and the generator neutral wires grounded?, $\{yes, no\}, reply$)
2. If $reply = yes$, then $n = 1$, else $n = 0$
3. End

Comments: By this d -function the customer specifies if the neutral wires are grounded. The range of options presented to the customer corresponds (after the conversion) to the values defined by the constraint $R_{\{n\}}$. Moreover, this d -function is defined for all the options.

f_b

Begin

4. **Ask** (Is the terminal block to be included? , {yes, no}, *reply*)
5. If *reply* = yes, then $b = 1$, else $b = 0$
6. End

Comments: By this d -function the customer expresses preference regarding the inclusion of an electrical interface for the ATS. The range of options presented to the customer corresponds (after the conversion) to the values defined by the constraint $R_{\{b\}}$. Moreover, this d -function is defined for all the options.

f_m

Begin

1. **Ask** (What is the material required for the enclosure? , {Metal, Plastic}, m)
- End

Comments: By means of this d -function the customer expresses his preference regarding the material for the ATS enclosure. The range of options presented to the customer corresponds to the values defined by the constraint $R_{\{b\}}$. Moreover, this d -function is defined for all the options.

$f_s(u, l)$

Begin

1. $s_v = u_v$
2. $AMP \leftarrow \{x : (s_v, x, _) \in \text{Table I-3} \wedge x \geq l_a\}$
3. If $AMP \neq \emptyset$ then
4. $s_a = \min AMP$
5. $s_d = z : (s_a, s_v, z) \in \text{Table I-3}$
6. $s = (s_v, s_a, s_d)$

End

Comments: The choice of the switch depends on the electrical characteristics of the utility and the load to which it will be connected. As it can be verified, line 1 incorporates the constraint $R_{\{u,s\}}$ and line 2 incorporates $R_{\{s\}}$ and $R_{\{l,s\}}$. Moreover, taking into account that the amperage the customer can assign to the load (l_a) has been restricted in accordance to Table I-3, and that the amperage s_a and voltage s_v are

related by this same table, we can conclude that $f_s(u, l)$ is defined for all allowed combination of values for the independent variables u and l .

$f_p(l, n)$

Begin

1. If $n = 0$ then $S_N = 1$, else $S_N = 0$
2. If $l_w = 2$ then $S_1 = 1 \wedge S_2 = 0 \wedge S_3 = 0$
3. If $l_w = 3$ then $S_1 = 1 \wedge S_2 = 1 \wedge S_3 = 0$
4. If $l_w = 4$ then $S_1 = 1 \wedge S_2 = 1 \wedge S_3 = 1$
5. $p = (S_N, S_1, S_2, S_3)$

End

Comments: The number of poles depends on the number of load wires that must be commuted. Because the procedure defining $f_p(l, n)$ satisfies the equation $S_N + S_1 + S_2 + S_3 = l_w - n$, it follows that the constraint $R_{\{n, l, p\}}$ is incorporated by it. Moreover, it is evident that, for every combination of values assigned to l and n , a value can be generated for p .

$f_e(b, s)$

Begin

1. If $b = 0$, then $e = \text{NA}$
2. Else $e = \min \{x : x \in \text{Table I-2} \wedge x \geq s_a\}$

End

Comments: The electrical characteristic of the terminal block must match the specifications of the switch. However, this is necessary only if it is included in the ATS configuration. As it can be verified, constraint $R_{\{e\}}$ is incorporated partly in line 1 and partly of line 2, and $R_{\{b, s, e\}}$ is incorporated in line 2. If $b = 0$, no matter what is the value assigned to s , $e = \text{NA}$. On the other hand, if $b = 1$, then, by construction, Tables I-2 and I-3 guarantee that for every value assigned to s_a there is an x such that $x \in \text{Table I-2}$ and $x \geq s_a$. Thus, for any combination of values of the independent variables b and s , a value can be generated for e .

$f_d(s, p)$

Begin

1. $P = S_N + S_1 + S_2 + S_3$
2. $(w', h', l') = s_d$
3. $d = (w, h, l): (w, h, l) \in \text{Table I-1} \wedge (\alpha + \beta + P \times w') \leq w \wedge (\varepsilon + h') \leq h \wedge (\delta + \gamma + l') \leq l$

End

Comments: The dimensions of the enclosure must be determined based on the dimensions of the components that go inside it, and on the necessary clearance required for their assembly and maintenance. As a simplification, in our example we take into account only the dimensions of the switch. In line 3, the enclosure is selected from Table I-1, thus satisfying constraint $R_{\{d\}}$. Since Table I-1 includes an enclosure that can handle the worst case, that is to say, a four pole 80A switch, there is always an enclosure that satisfies the condition of constraint $R_{\{s,p,d\}}$. Hence, $f_d(s, p)$ is defined for all combinations of values assigned to the independent variables.

$f_m(d)$

Begin

1. Find the first $x : (d, x) \in \text{Table I-1}$
2. $m = x$

End

Comments: This is an alternative d -function attached to variable m . While the input d -function f_m depends on the customer, $f_m(d)$ generates the value for m from Table I-1. Since the material assigned to m is one of the available in Table I-1, it implies that the d -function incorporates constraint $R_{\{m\}}$. Moreover, since the values for variables d and m are related by Table I-1, the constraint $R_{\{m,d\}}$ is also incorporated. Furthermore, since the value assigned to d by $f_d(s, p)$ is one of the sizes in Table I-1, there is always an associated material that can be assigned to m . Therefore, the procedure above always generates a value to m , which implies that the d -function is defined for all the values assigned to d .

$f_c(u, g)$

Begin

1. $r = g_w$
2. If $r \geq 8$, then $DS8 = 1 \wedge r = r - 8$, else $DS8 = 0$

3. If $r \geq 4$, then $DS7 = 1 \wedge r = r - 4$, else $DS7 = 0$
4. If $r \geq 2$, then $DS6 = 1 \wedge r = r - 2$, else $DS6 = 0$
5. If $r \geq 1$, then $DS5 = 1$, else $DS5 = 0$
6. If $u_p = 3$, then $DS4 = 1$, else $DS4 = 0$
7. If $u_f = 60$, then $DS3 = 1$, else $DS3 = 0$
8. If $u_v = 220$, then $DS2 = 1$, else $DS2 = 0$
9. If $g_m = 1$, then $DS1 = 1$, else $DS1 = 0$
10. $c = (DS1, DS2, DS3, DS4, DS5, DS6, DS7, DS8)$

End

Comments: The configuration of the DIP switch depends on the values of the related variables. Lines 1-5 convert the waiting time (a decimal number) into a binary number and line 9 specify if the engine has to be monitored. These lines correspond to constraint $R_{\{g,c\}}$. Lines 6-8 specify the electrical characteristics of the utility. They correspond to constraint $R_{\{u,c\}}$. Thus, it can be concluded that these two constraints are incorporated in $f_c(u, g)$. Moreover, for every combination of values assigned to the independent variables u and g , the procedure generates a value to variable c , which implies that $f_c(u, g)$ is defined for all values assigned to the independent variables.

$f_a(c)$

Begin

1. If $DS1 = 1$, then $a = 1$, else $a = 0$

End

Comments: The inclusion of the buzzer depends on the necessity of monitoring the engine-generator group. Clearly, this d -function incorporates the constraint $R_{\{c,a\}}$, and it is defined for all values of the independent variable c .

$f_h(a)$

Begin

1. If $a = 1$, then $h = 1$, else $h = 0$

End

Comments: The requirement of a hole on the enclosure depends on the inclusion of the buzzer into the custom ATS. Clearly, this d -function incorporates the constraint $R_{\{a,h\}}$, and it is defined for all values of the independent variable a .

$f_w(a, b, p, s)$

Begin

1. $WIRING = WIRE_TEMPLATE$
2. If $a = 0$, then $WIRING = WIRING - \{(B_1, B_2)\}$
3. If $b = 0$, then $WIRING = WIRING - \left\{ \begin{array}{l} (O_1, O_2), (C_1, C_2), (SU_1, U_1), (SG_1, G_1), \\ (SL_1, L_1), (SU_2, U_2), (SL_2, G_2), (SL_2, L_2), \\ (SU_3, U_3), (SG_3, G_3), (SL_3, L_3), (SU_n, U_n), \\ (SG_n, G_n), (SL_n, L_n) \end{array} \right\}$
4. If $S_2 = 0$, then $WIRING = WIRING - \{(SU_2, U_2), (SU_2, M_2), (SG_2, G_2), (SL_2, L_2)\}$
5. If $S_3 = 0$, then $WIRING = WIRING - \{(SU_3, U_3), (SU_3, M_3), (SG_3, G_3), (SL_3, L_3)\}$
6. If $S_N = 0$, then $WIRING = WIRING - \{(SU_N, U_N), (SU_N, M_N), (SG_N, G_N), (SL_N, L_N)\}$
7. $A = \min \{y : (y, x) \in \text{Table I-4} \wedge x \geq s_a\}$
8. $w = (WIRING, A)$

End

Comments: The wiring can be configured only after the components going inside the ATS are specified, to determine which terminals must be connected and what currents must be supported. Line 2 satisfies constraint $R_{\{a,w\}}$, line 3 satisfies constraint $R_{\{b,w\}}$, lines 4-6 satisfy constraint $R_{\{p,w\}}$, and line 7 satisfies constraints $R_{\{w\}}$ and $R_{\{s,w\}}$. Each of the lines, from 2 to 6, defines independently a portion of the wiring. Moreover, Table I-4 contains a wire that can stand any of the amperages of the available switches. Therefore, no matter what is the combination of values of the independent variables, a value for the variable w can always be generated.

I.4 Instantiation patterns

In Section 4.1.1 of the main body of this thesis, we have concluded that if some variable is attached to more than one d -function, there will be more than one instantiation pattern in F , each one giving rise to a correspondent instantiation graph. The d -functions specified in the preceding section can be gathered to form two instantiation patterns, shown as columns P_1 and P_2 in the Table I-5. Note that, except for f_m and $f_m(d)$, all other d -functions are contained in both instantiation patterns. The instantiation graphs that come out when these patterns are followed during the instantiation process are shown in Figure I-5, labelled G_1 and G_2 , respectively.

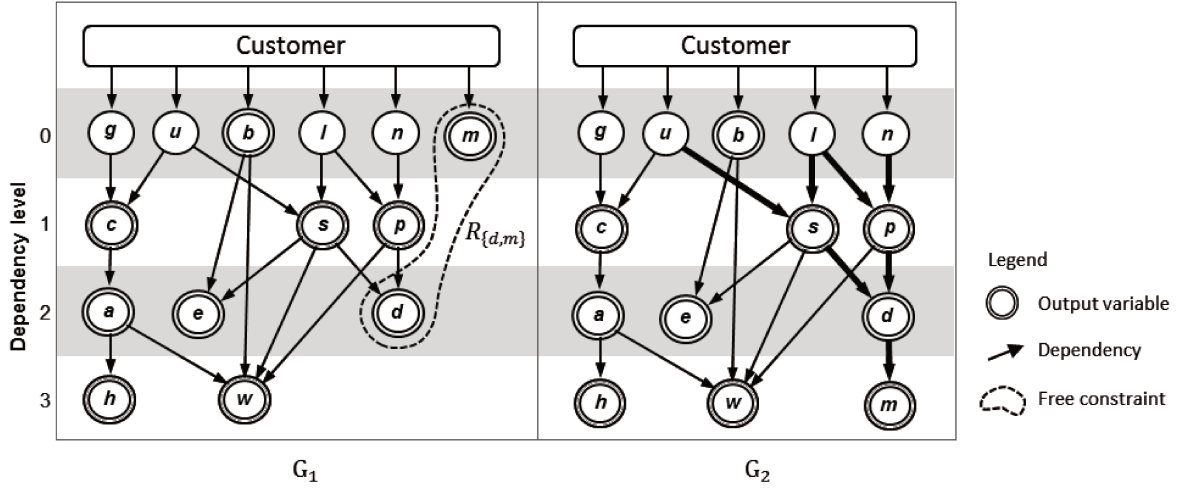


Figure I-5. The instantiation graphs for the ATS product family example

Based on our comments following the specification of the d -functions, we can conclude that the Consistency Condition 1 (defined in Section 4.1.2 of the main body of this thesis) is satisfied by all d -functions since all of them are defined for all the combinations of values assigned to their independent variables. However, the instantiation pattern P_1 does not satisfy the Consistency Condition 2. This can be seen in Table I-5, which summarizes the incorporation of constraints by the d -functions. As indicated by the highlighted rows, the d -function f_m does not incorporate the constraint $R_{\{m,d\}}$ and $f_m(d)$ is the only d -function to do that. Moreover, the latter d -function is not included in the instantiation pattern P_1 . On the other hand, all constraints are incorporated in one of the d -functions belonging to the instantiation pattern P_2 . This proves that P_2 is the only instantiation pattern of the CN-F model for the ATS product family that satisfies the second consistency conditions.

Table I-5. Summary of the relations between constraints, d -functions and instantiation patterns of the CN-F model for the ATS product family

Constraints	d -functions	P_1	P_2
$R_{\{u\}}$	f_u	yes	yes
$R_{\{g\}}$	f_g	yes	yes
$R_{\{l\}}$	f_l	yes	yes
$R_{\{n\}}$	f_n	yes	yes
$R_{\{m\}}$	f_m	yes	no
$R_{\{m,d\}}$	$f_m(d)$	no	yes
$R_{\{b\}}$	f_b	yes	yes
$R_{\{e\}}$	$f_e(b, s)$	yes	yes
$R_{\{c,a\}}$	$f_a(c)$	yes	yes
$R_{\{a,h\}}$	$f_h(a)$	yes	yes
$R_{\{g,c\}}$	$f_c(u, g)$	yes	yes
$R_{\{u,c\}}$			
$R_{\{s\}}$	$f_s(u, l)$	yes	yes
$R_{\{u,s\}}$			
$R_{\{l,s\}}$			
$R_{\{w\}}$	$f_w(a, b, p, s)$	yes	yes
$R_{\{a,w\}}$			
$R_{\{b,w\}}$			
$R_{\{p,w\}}$			
$R_{\{s,w\}}$			
$R_{\{n,l,p\}}$	$f_p(l, n)$	yes	yes
$R_{\{b,s,e\}}$	$f_e(b, s)$	yes	yes
$R_{\{d\}}$	$f_d(s, p)$	yes	yes
$R_{\{s,p,d\}}$			

To avoid having to deal with inconsistencies, we can simply remove f_m from F , the source of inconsistencies in our CN-F modelling of the ATS example. The remaining set of d -functions coincides to P_2 and, therefore, the new CN-F model satisfies both consistency conditions. As it was discussed in Section 4.3 of the main body of this thesis, the inconvenience of removing f_m is that we lose possible solutions. The implication is that some of the members of the ATS are not reached by the customization method. In that section, we also present an alternative way to overcome the inconsistencies associated to the instantiation pattern P_1 , by defining a new input d -function $f'_m(d)$ that presents to the customer only options of materials that are consistent with the value already assigned to the variable d and replacing it by f_m . In case no solution is lost.

I.5 The instantiation algorithm

The instantiation algorithm for the CN-F model was introduced in Figure 12, Section 4.1, in the main body of this thesis. This algorithm was instrumental in the definition of the consistency conditions. It is capable of identifying inconsistencies of type I and II during the instantiation process. However, if the CN-F model satisfies both inconsistency conditions, that algorithm can be simplified by removing the lines 4-12 related to the checking of inconsistencies. The resulting algorithm is shown in Figure I-6. This algorithm is quite straightforward and admits no backtracking. Actually, as we shall see in Appendix II, this algorithm can be implemented as a dataflow program, by the concatenation of the d -functions of the CN-F model.

Instantiation algorithm:

Begin

1. Following the order in F , remove the first f_x which is enabled
2. Generate a value to x using f_x and remove from F all design functions attached to x
3. If $F \neq \emptyset$
4. Then go to 1
5. Else go to End and return "Success"

End

Figure I-6. A simplified instantiation algorithm for CN-F models that satisfy Consistency Condition 1 and 2

Appendix II

Implementation of the ATS Customisation Process

In this appendix, we present the implementation of the customisation process of the ATS product family using the programming language LabVIEW. However, this implementation covers only the first stage of this process, to find solutions to the CN-F model. Appendix II is divided into two parts. In Part I, we present the implementation of the d -functions, the control structure and the interface of the system to display the solutions of the customisation process. This implementation follows the modelling presented in Appendix I. In Part II, we use this program to run the case introduced in Section 4.3 of the main body of the thesis.

Part I – The implementation

LabView is a graphical programming language from National Instruments. One of the main reasons for choosing this language for the implementation of our approach is the fact that LabVIEW is based on the principle of dataflow, in which the functions are executed only after all the data required is available. This principle matches quite well with the concept of instantiation patterns and the algorithm for the instantiation process, thus facilitating the programming of the configurator. Actually, when the CN-F model satisfies the inconsistency conditions, the resulting process turns out to be a dataflow program, as can be verified in Figure II-15. Another major advantage of LabVIEW is the level of modularity that can be obtained. The resulting software program is such that any d -function can be improved without affecting the other functions or the control structure.

In what follows, we first present the implementation of each d -function based on the specifications in Subsection I.3.3 of Appendix I. Note that the implementation of the d -functions is composed of at least two views. This is because they are implemented using the Case Structure of LabVIEW, with the state True indicating the inclusion of the d -function in the set F and the state False

indicating its removal from F , to implement line 1 of the instantiation algorithm. However, if another Case Structure is also used inside, there will be more than two views. Note also that some of the d -functions will read data from .txt files (for example, see Figure II-6). Although these files are part of the implementation, they have not been included explicitly in this appendix since they correspond strictly to the tables of Appendix I.

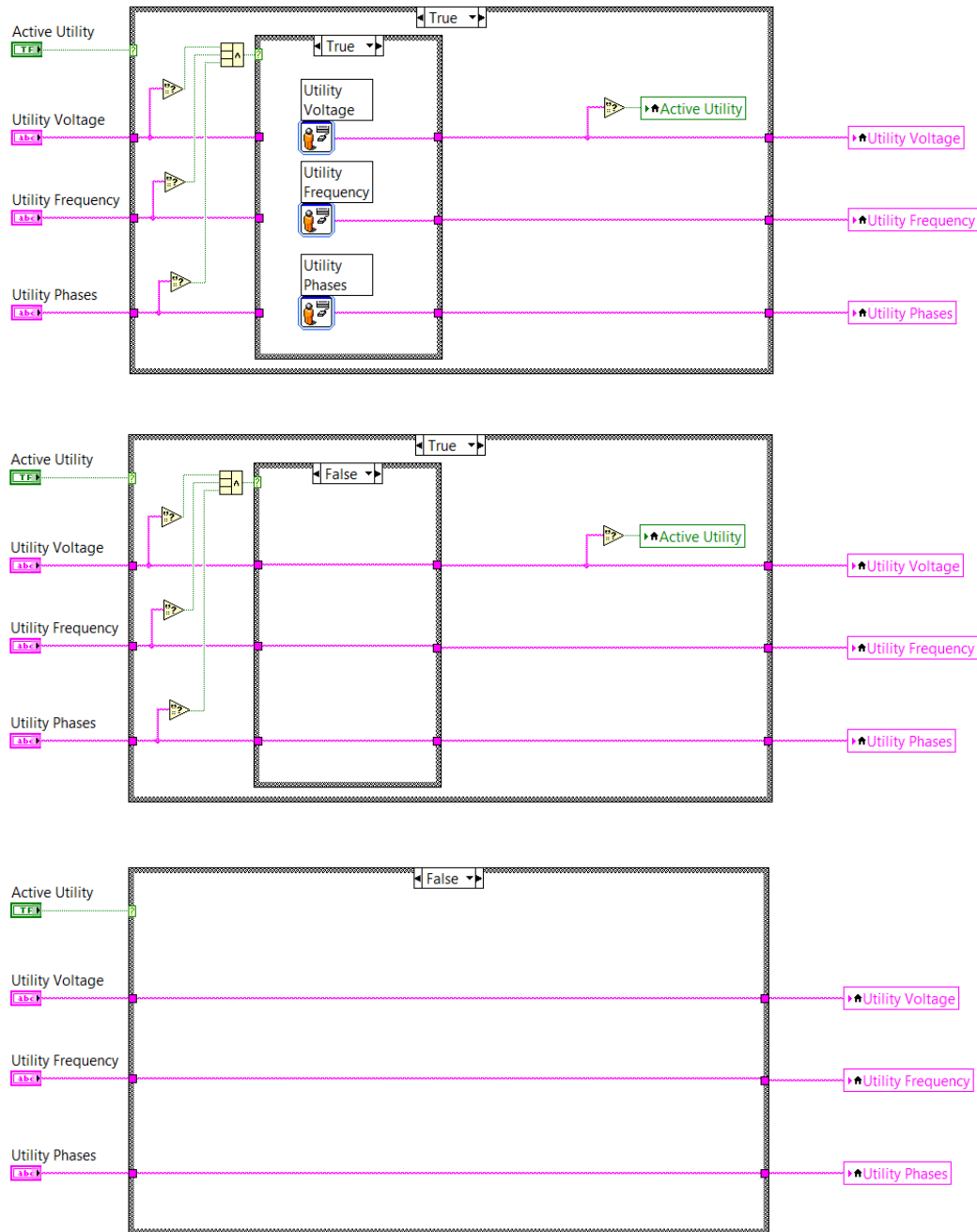


Figure II-1. The d -function attached to the variable *Utility*

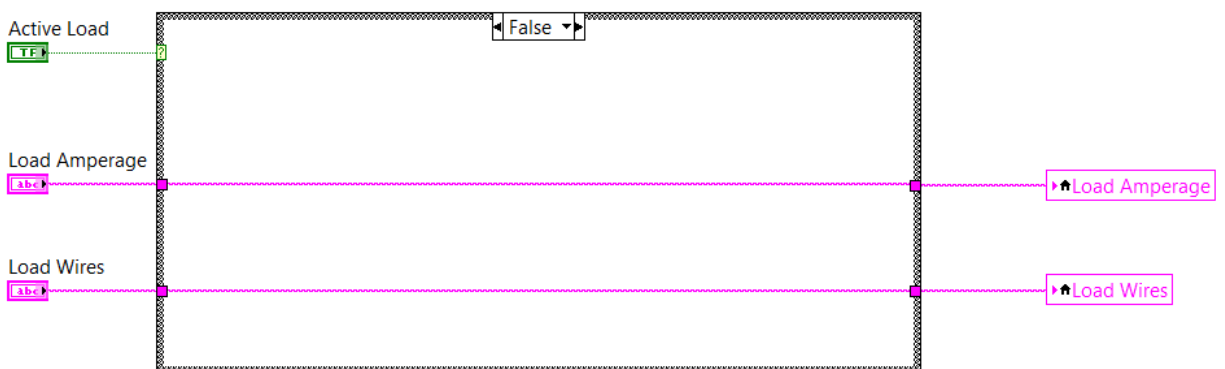
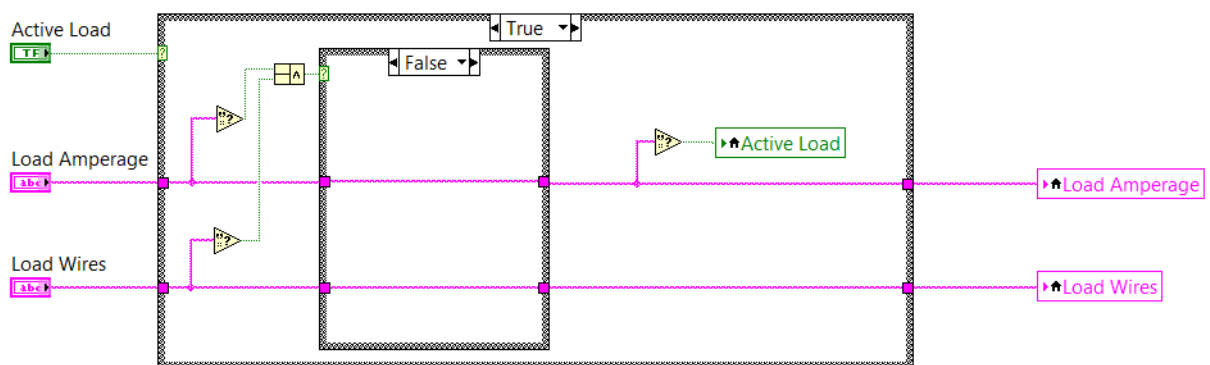
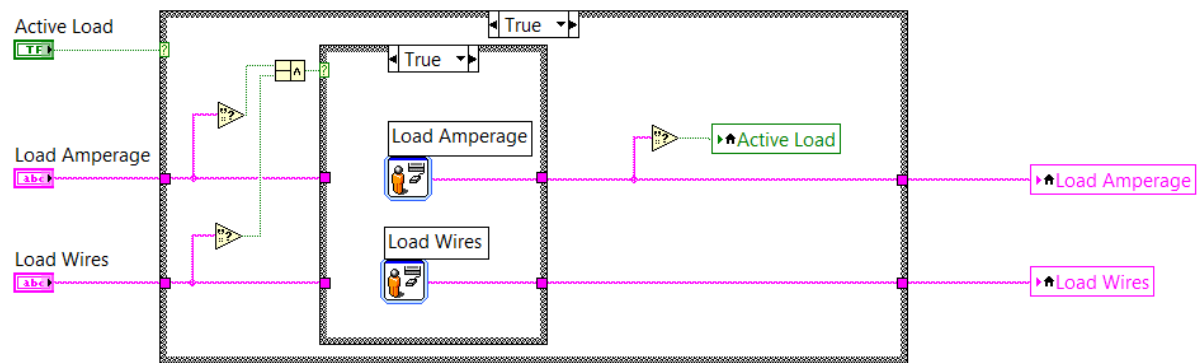


Figure II-3. The *d*-function attached to the variable *Emergency load*

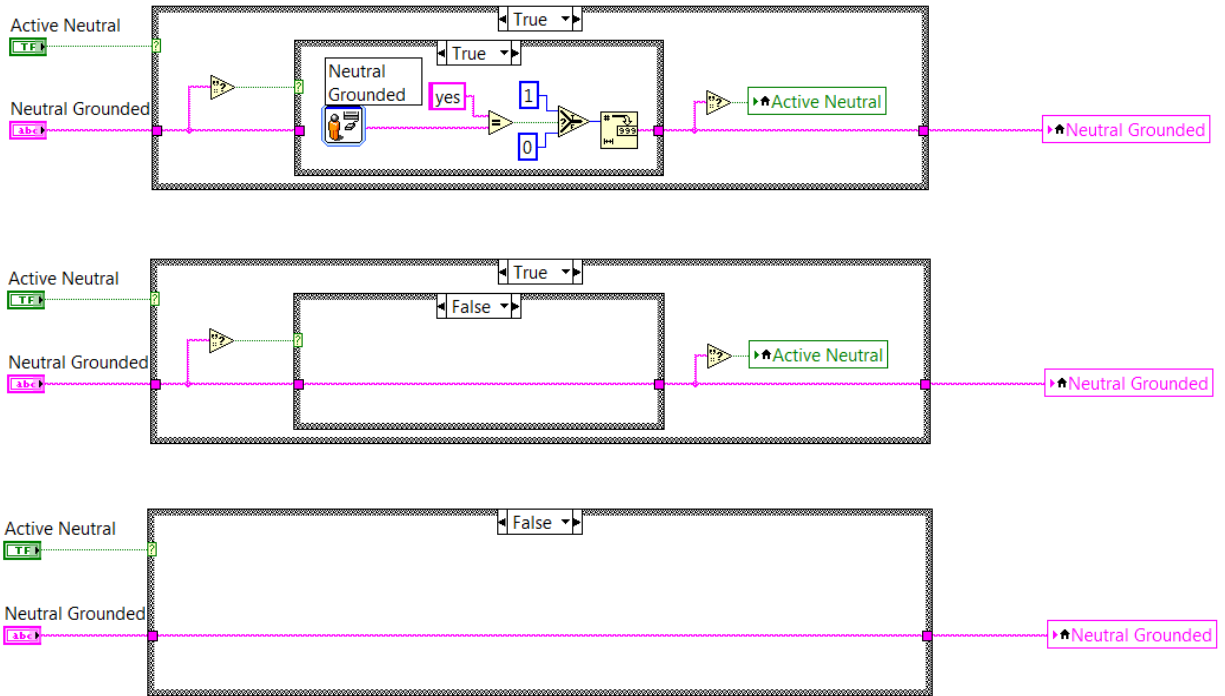


Figure II-4. The *d*-function attached to the variable *Neutral grounding*

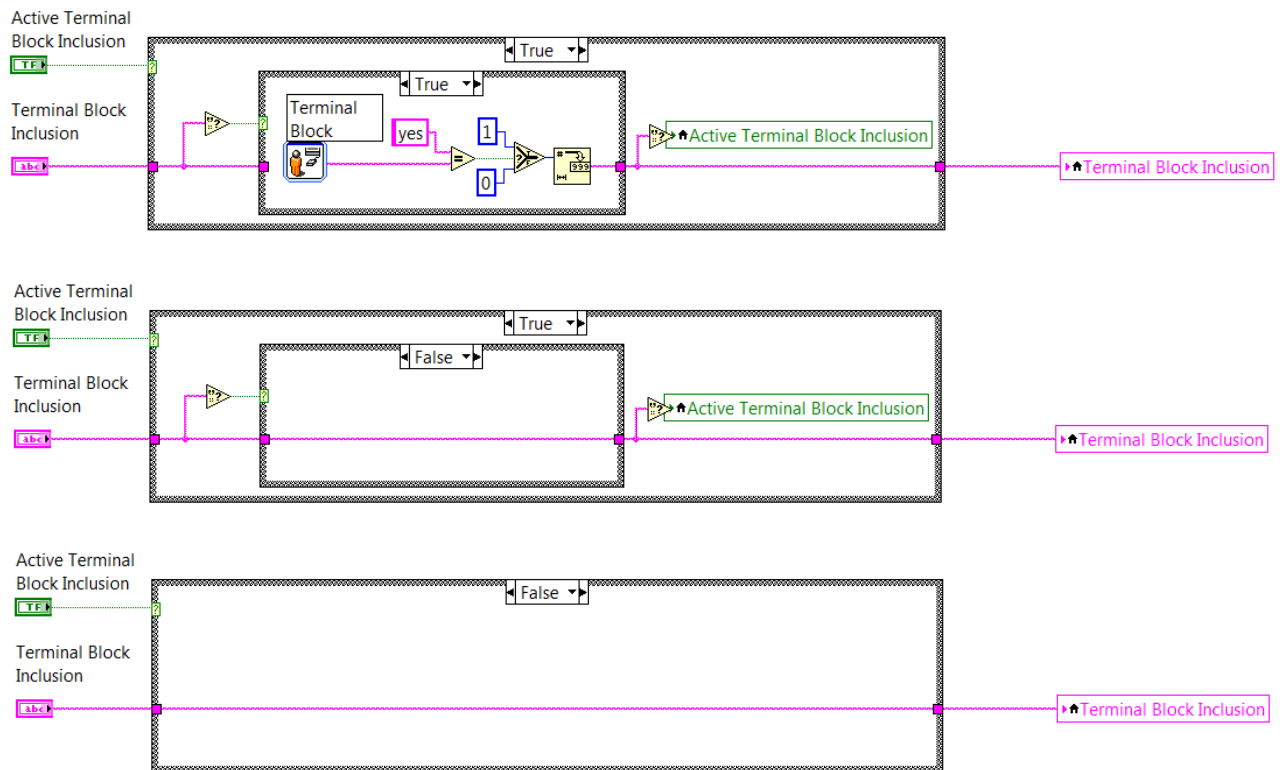


Figure II-5. The *d*-function attached to the variable *Terminal block inclusion*

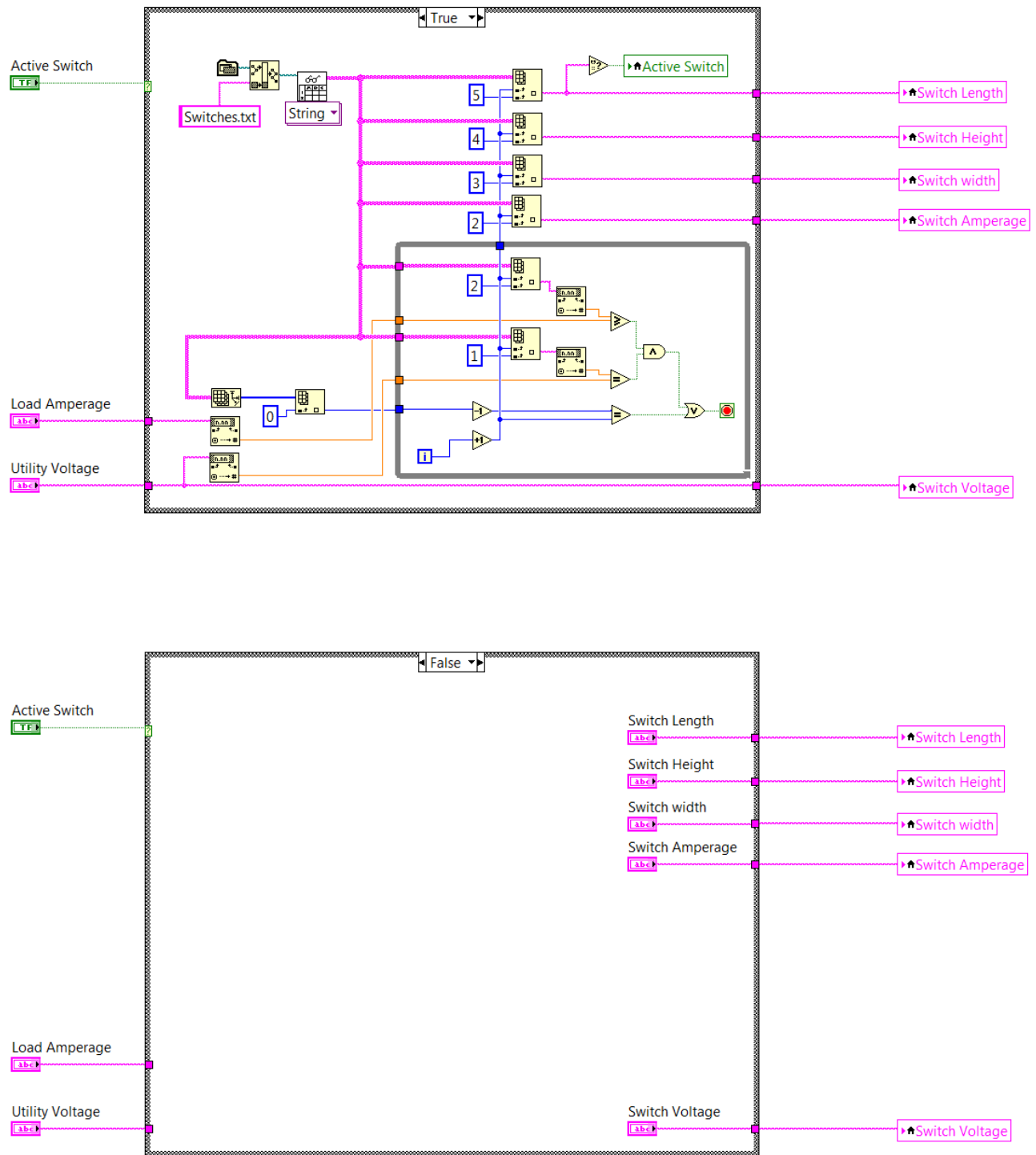


Figure II-6. The *d*-function attached to the variable *Switch specification*

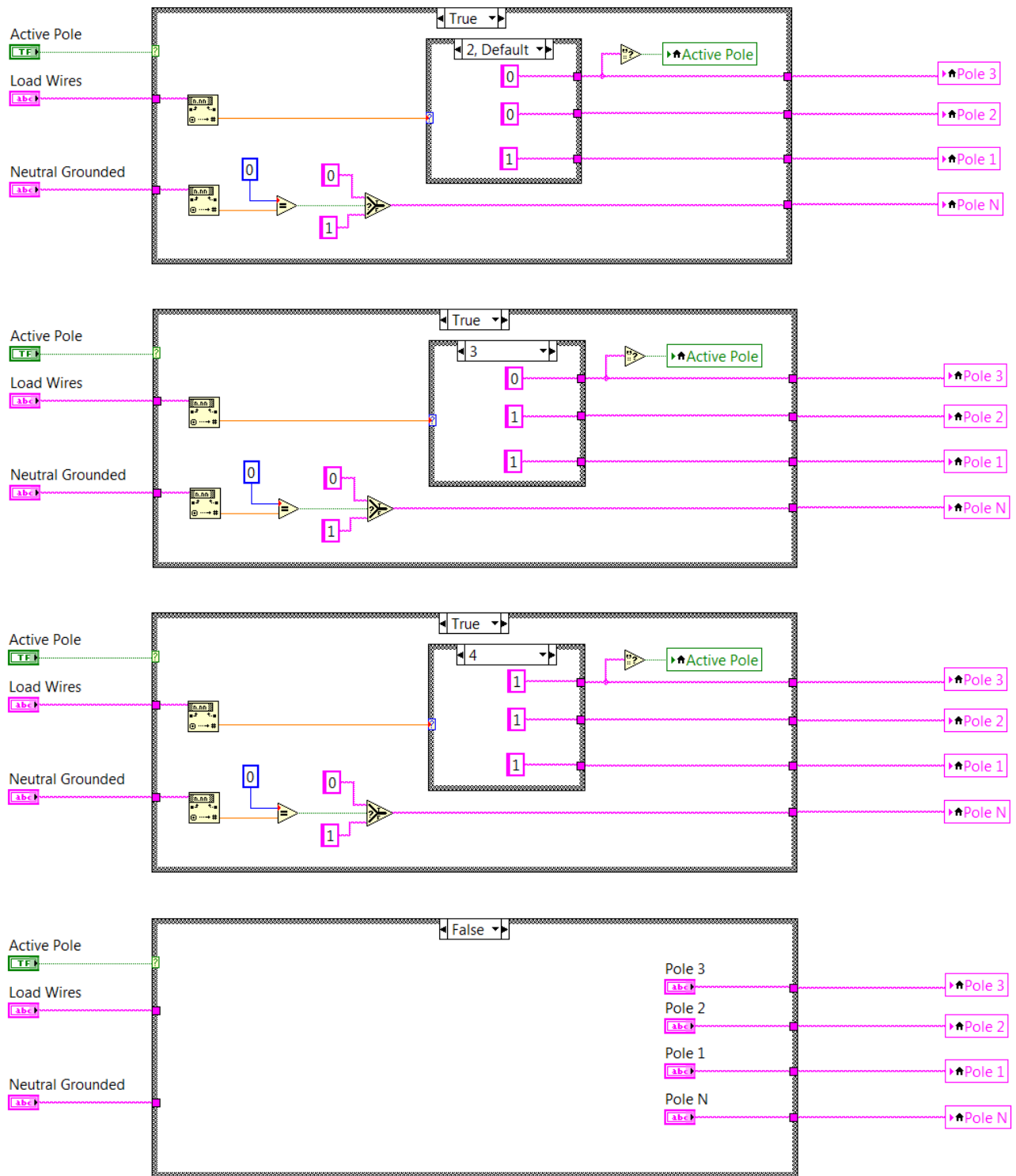


Figure II-7. The *d*-function attached to the variable *Switch poles*

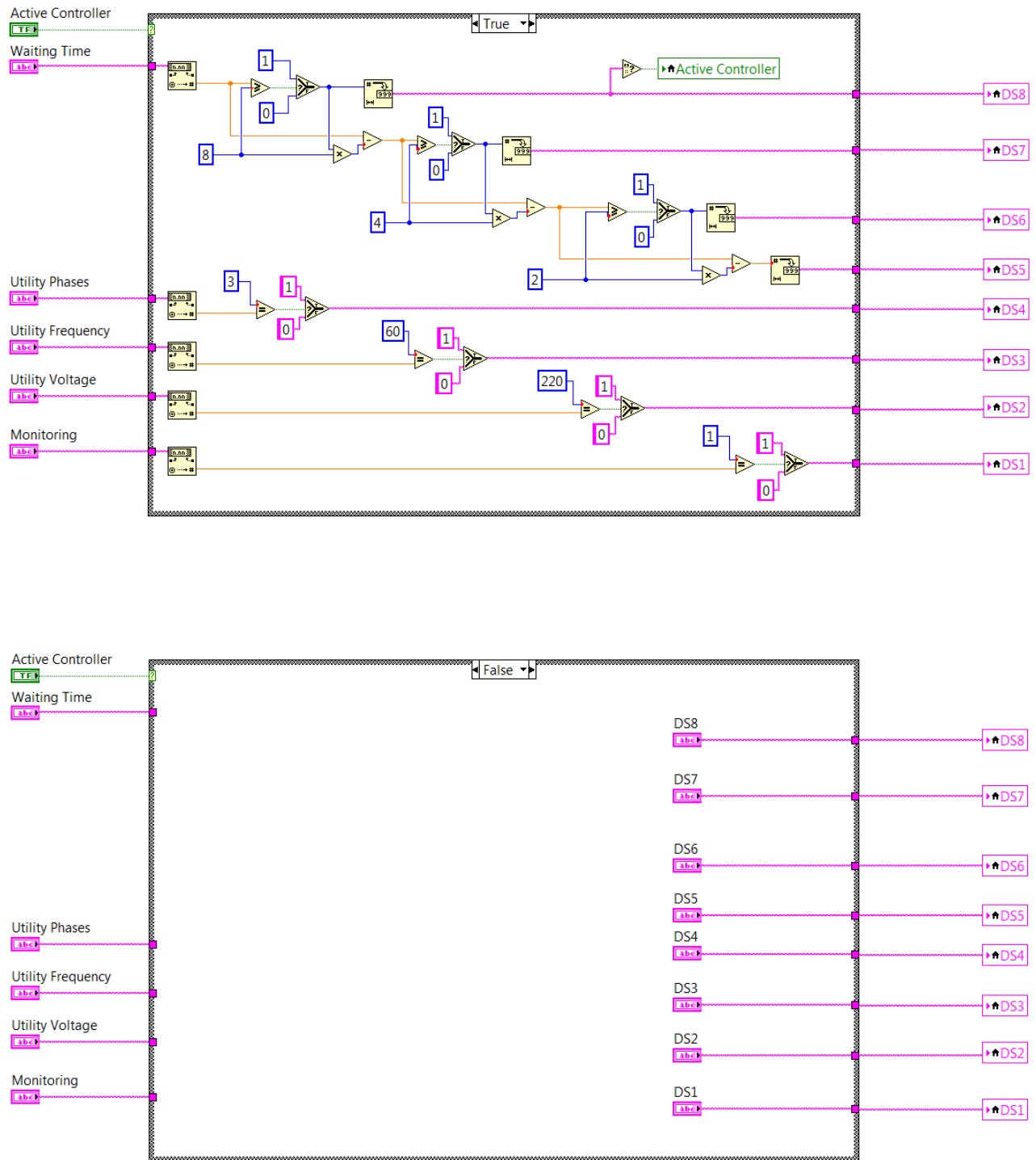


Figure II-8. The *d*-function attached to the variable *Control board configuration*

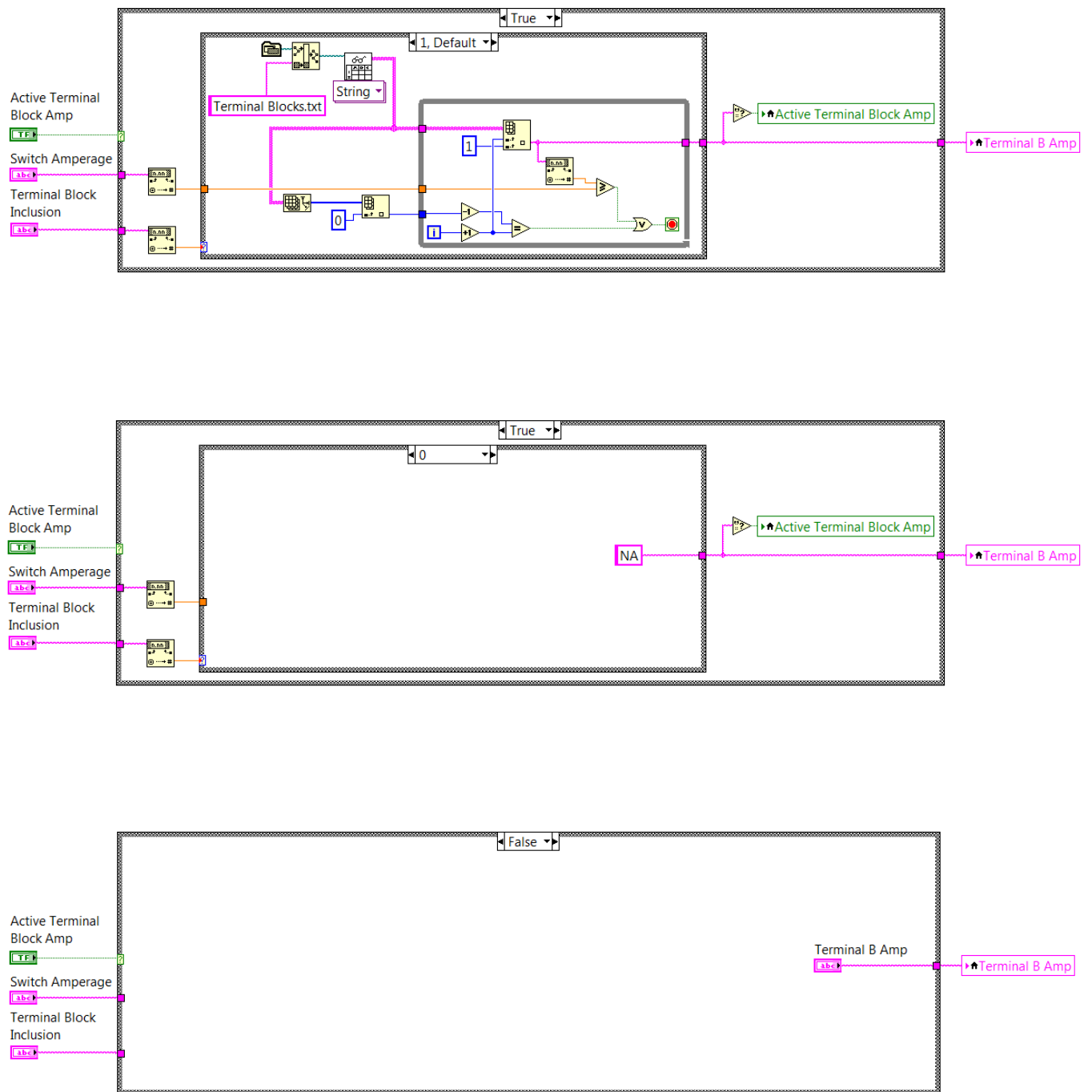


Figure II-9. The *d*-function attached to the variable *Terminal block amperage*

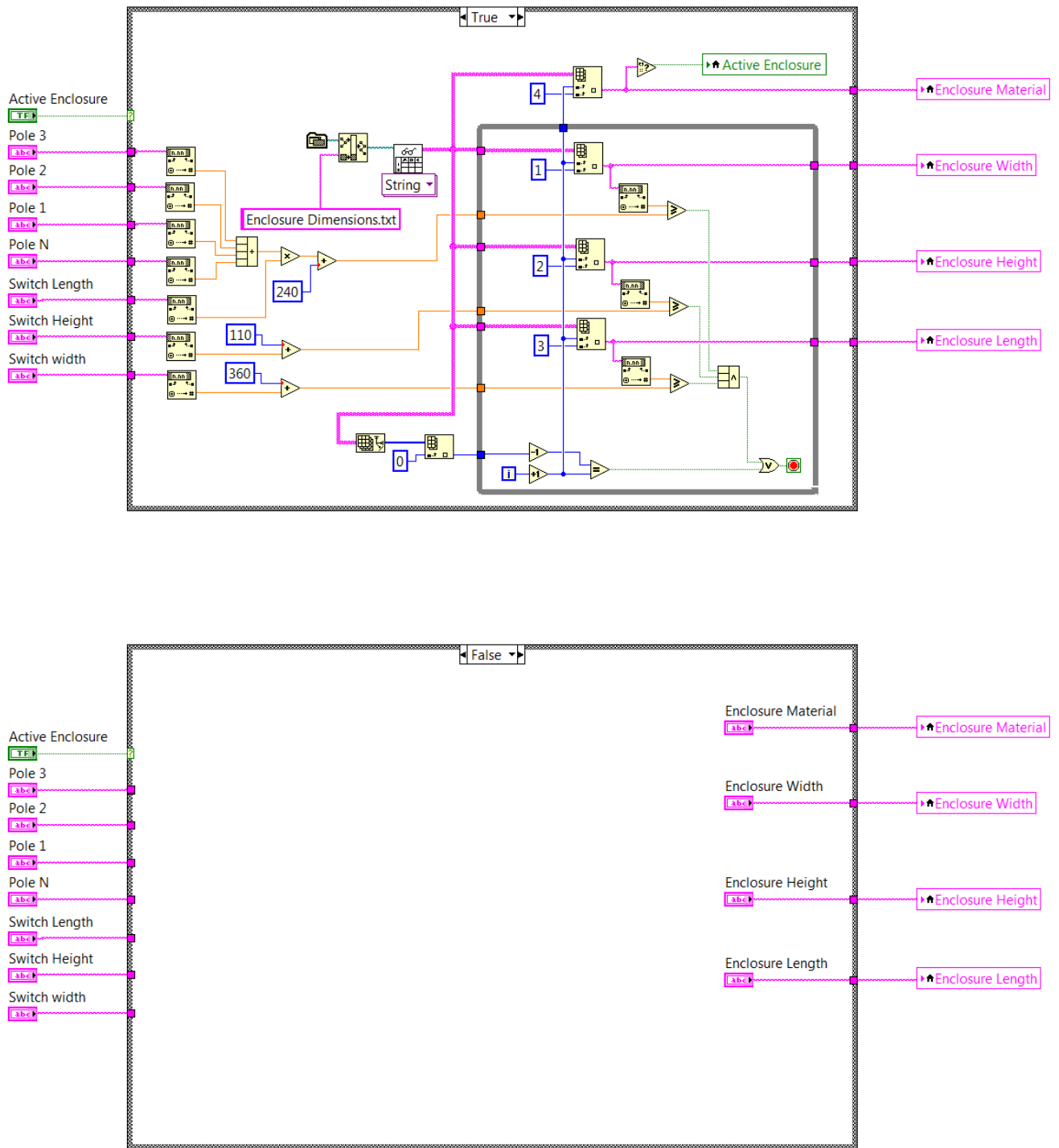


Figure II-10. The d -function attached to the composite variable *Enclosure*

Note that, for convenience, the d -function shown in Figure II-10 combines the specification of the d -functions $f_m(d)$ and $f_d(s,p)$ to generate the values for the variables *Enclosure material* and *Enclosure dimensions*.

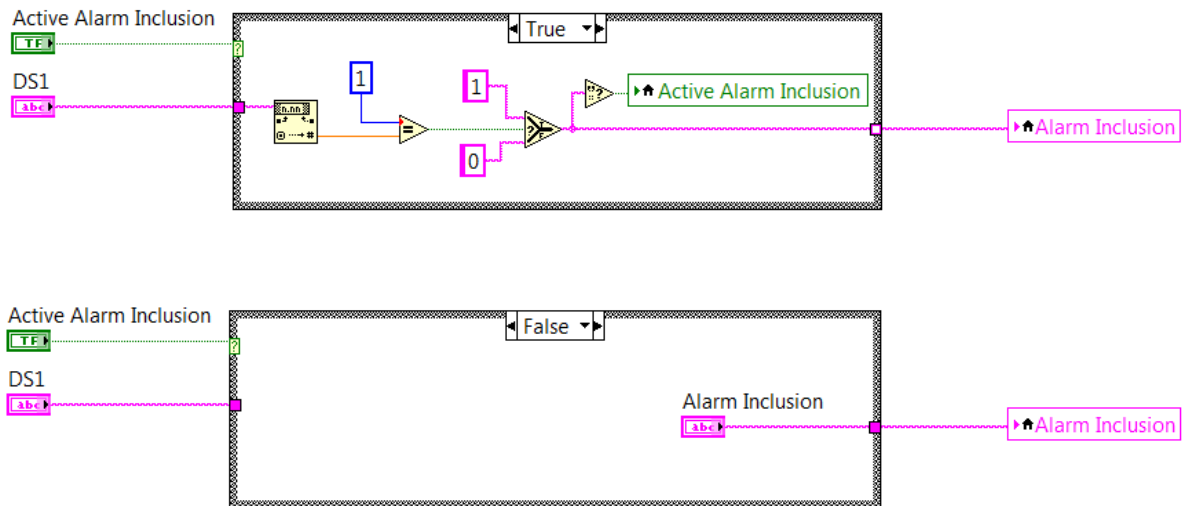


Figure II-11. The d -function attached to the variable *Buzzer inclusion*

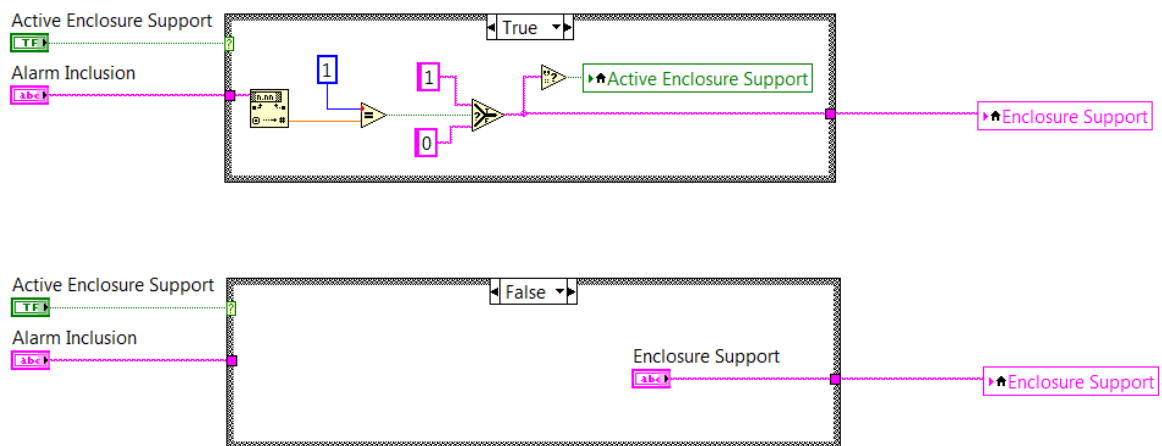


Figure II-12. The d -function attached to the variable *Enclosure hole*

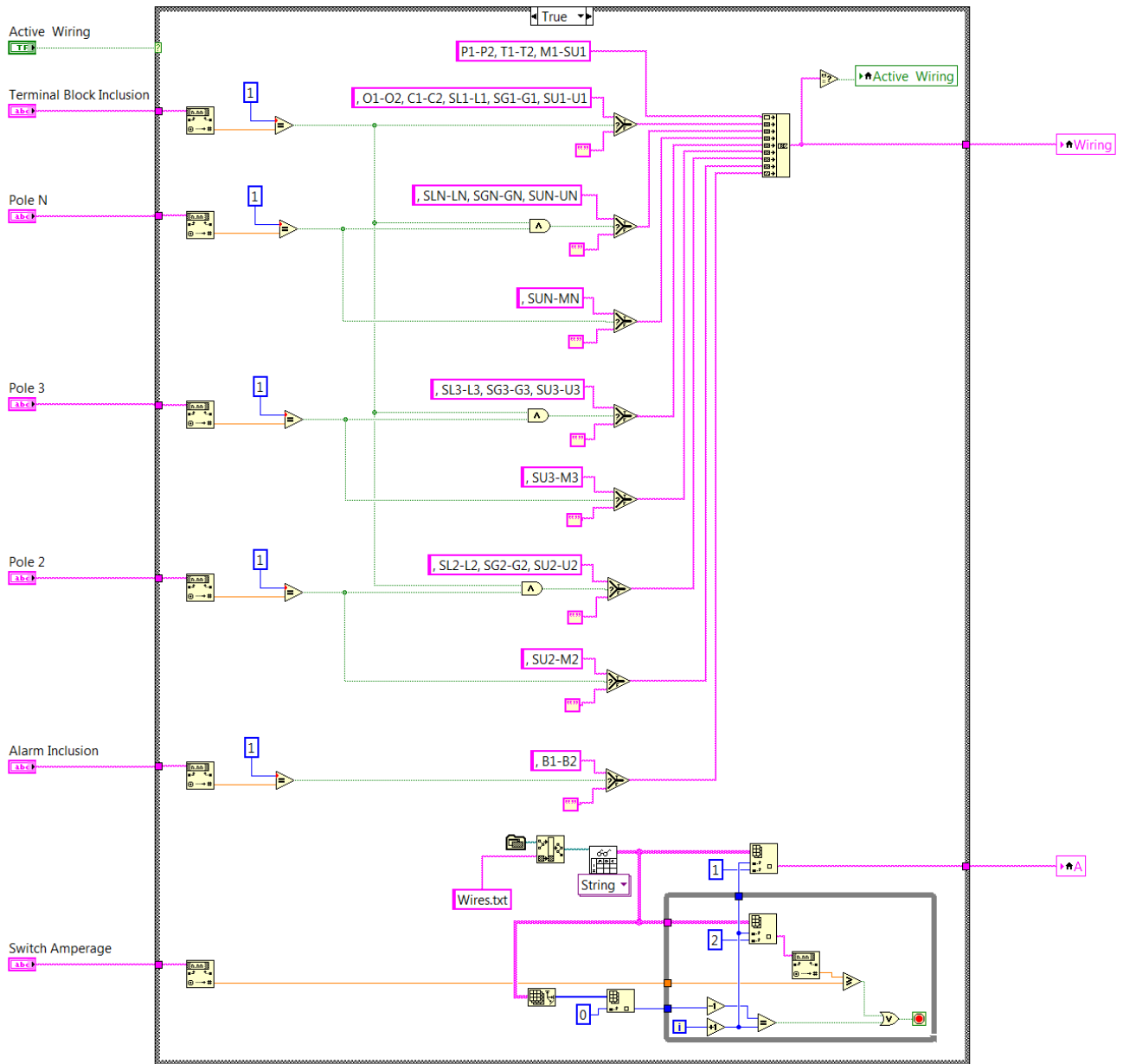


Figure II-13. One view of the d -function attached to the variable *Wiring configuration*

Note that the implementation of this d -function does not correspond exactly to the specification presented in Appendix I. Instead of beginning with the wire template and removing the unused wires, we start with the set of common wires and add the necessary optional wires.

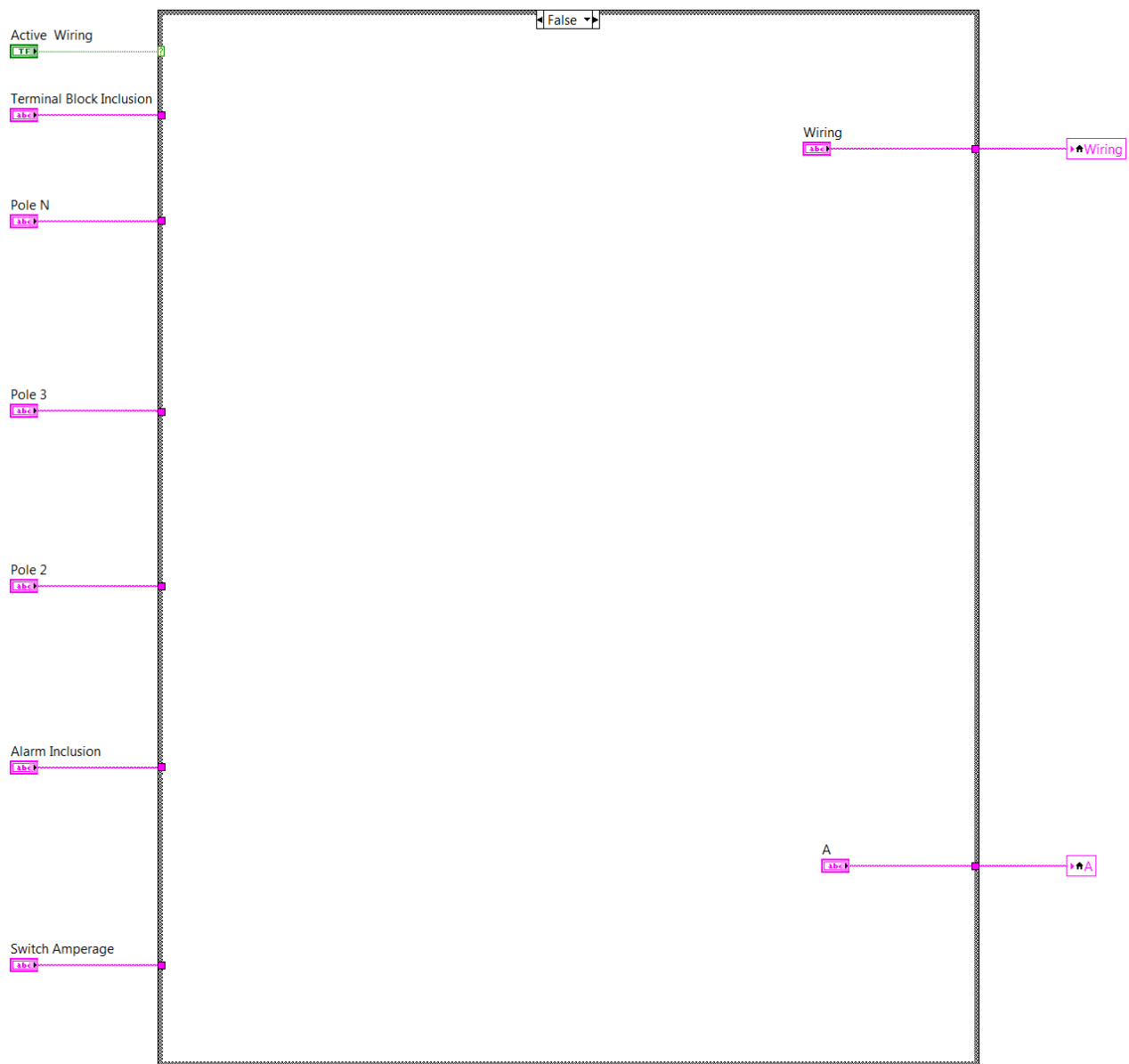


Figure II-14. Second view of the d -function attached to the variable *Wiring configuration*

The implementation of the control structure for the instantiation algorithm is shown in Figure II-15. Note that, the instantiation algorithm presented in Figure I-6 has an iterative structure where only one d -function is executed during each iteration. However, in this implementation, we prefer to stress the dataflow character of the instantiation process. In this setting, all d -functions are executed in only one iteration. For the sake of visibility, each of the d -functions presented in the preceding figures appears as a SubVI (a kind of subroutine in LabVIEW) within the While Loop structure of LabVIEW. They are represented as boxes in the diagonal of the While Loop and numbered from F1 to F13. The input lines connecting the bottom of a d -function from the outputs of lower level d -functions indicate the dependencies between the d -functions.

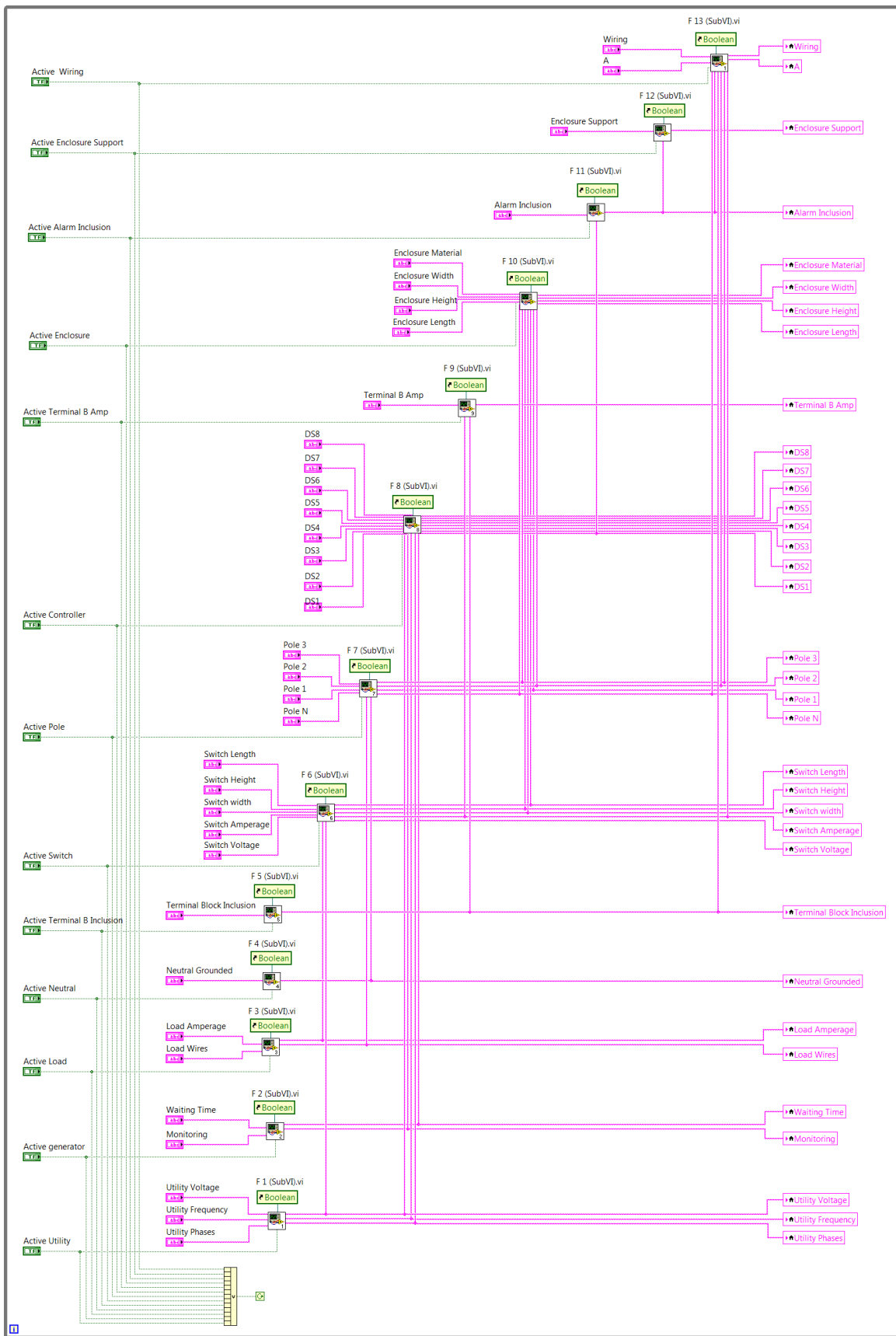


Figure II-15. Implementation of the control structure of the instantiation algorithm

Initially, all the variables of the type *Active* “X” are set to True, making the *d*-functions available for execution. As the *d*-functions are executed, these variables are set to False, to inactivate the *d*-function. Because of the dependencies, this process begins with the *d*-functions at the bottom of the iterative structure, which corresponds to the input variables. When all the *d*-functions are disabled the program ends, and a solution has been found.

DESIGN FUNCTIONS	VARIABLES							
Active Wiring	Wiring							
Active Enclosure Support	Enclosure Support							
Active Alarm Inclusion	Alarm Inclusion							
Active Enclosure	Enclosure Width	Enclosure Height	Enclosure Length	Enclosure Material				
Active Terminal Block Amp	Terminal Block Amp							
Active Controller	DS1	DS2	DS3	DS4	DS5	DS6	DS7	DS8
Active Pole	Pole 3	Pole 2	Pole 1	Pole N				
Active Switch	Switch Voltage	Switch Amperage	Switch width	Switch Height	Switch Length			
Active Terminal Block Inclusion	Terminal Block Inclusion							
Active Neutral	Neutral Grounded							
Active Load	Load Amperage	Load Wires						
Active Generator	Waiting Time	Monitoring						
Active Utility	Utility Voltage	Utility Frequency	Utility Phases					

Legend
 Input variable
 Output variable
 Function enabled
 Function disabled

Figure II-16. The Front Panel of LabVIEW showing all the variables used in the ATS configurator

Figure II-16 shows the Front Panel of LabVIEW with all the variables used in the program. At the left, it can be seen the variables of the type *Active* “X”, all of them initially set to True (buttons of the corresponding switches turned to the right), indicating that the *d*-functions are available to generate the values for the variables they are attached. At the right, it can be seen all the variables related to the ATS product family, and the corresponding fields to be filled with the values generated by the correspondent *d*-functions, as they are executed.

Part II – An example of the customisation process of the ATS product family

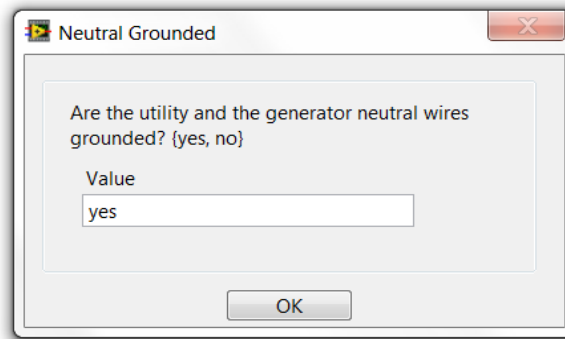
The customisation process starts with the configurator asking the customer to enter the requirements of

the ATS through a sequence of windows. This happens while the input *d*-functions are being executed. For our prototype system, it has been assumed that the customers answer all the questions by choosing values within the limits or options set by the questions. In what follows, the input windows that are prompted by the same input *d*-function are grouped together. Since all the inputs are independent of each other, there is no restriction in the order the windows are presented to the customer. The answers for the questions are those values of the example in the Section 4.3 of the main body of the thesis.

The figure displays three separate input windows, each with a title bar, a question, a value input field, and an OK button.

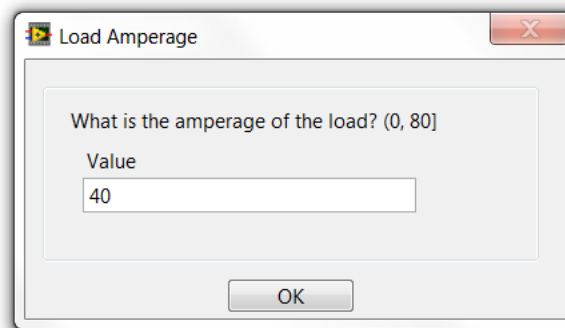
- Utility Voltage:** The question is "What is the utility voltage? {110, 220}". The value input field contains "110".
- Utility Phases:** The question is "What is the number of phases of the utility? {1, 3}". The value input field contains "1".
- Utility Frequency:** The question is "What is the utility frequency? {50, 60}". The value input field contains "60".

Figure II-17. Inputs regarding the *Utility*

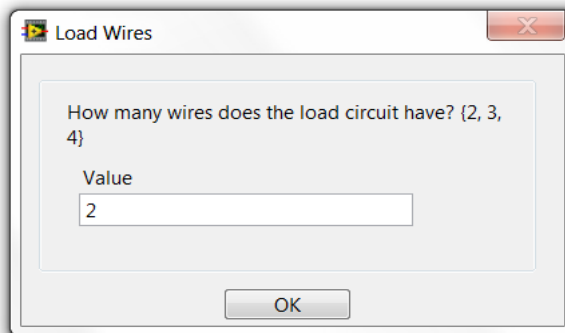


A dialog box titled "Neutral Grounded" with a close button (X) in the top right corner. The main text asks: "Are the utility and the generator neutral wires grounded? {yes, no}". Below this, there is a label "Value" and a text input field containing the word "yes". At the bottom center is an "OK" button.

Figure II-18. Input regarding the *Neutral grounding*

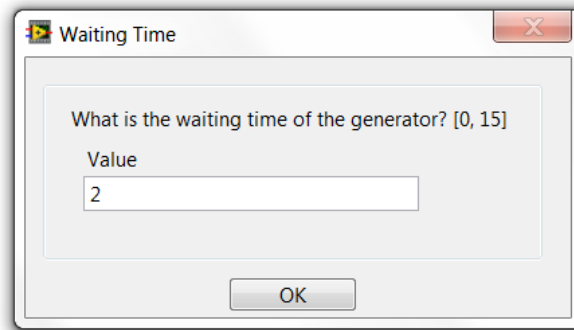


A dialog box titled "Load Amperage" with a close button (X) in the top right corner. The main text asks: "What is the amperage of the load? (0, 80]". Below this, there is a label "Value" and a text input field containing the number "40". At the bottom center is an "OK" button.

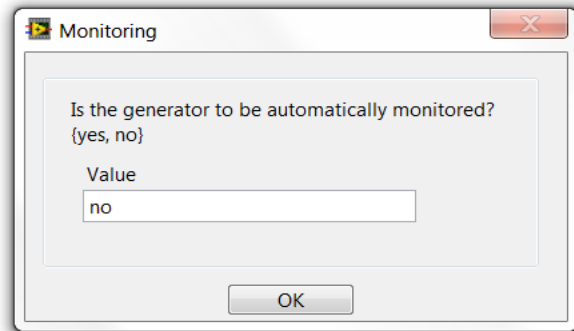


A dialog box titled "Load Wires" with a close button (X) in the top right corner. The main text asks: "How many wires does the load circuit have? {2, 3, 4}". Below this, there is a label "Value" and a text input field containing the number "2". At the bottom center is an "OK" button.

Figure II-19. Inputs regarding the *Emergency load*

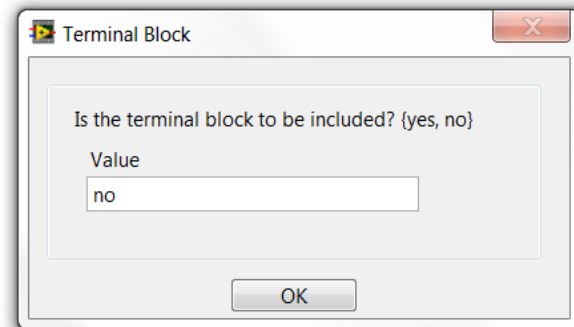


A dialog box titled "Waiting Time" with a close button (X) in the top right corner. The main text asks "What is the waiting time of the generator? [0, 15]". Below this, the label "Value" is followed by a text input field containing the number "2". At the bottom center is an "OK" button.



A dialog box titled "Monitoring" with a close button (X) in the top right corner. The main text asks "Is the generator to be automatically monitored? {yes, no}". Below this, the label "Value" is followed by a text input field containing the word "no". At the bottom center is an "OK" button.

Figure II-20. Inputs regarding the *Motor-generator group*



A dialog box titled "Terminal Block" with a close button (X) in the top right corner. The main text asks "Is the terminal block to be included? {yes, no}". Below this, the label "Value" is followed by a text input field containing the word "no". At the bottom center is an "OK" button.

Figure II-21. Input regarding the *Terminal block inclusion*

DESIGN FUNCTIONS	VARIABLES									
Active Wiring	Wiring P1-P2, T1-T2, M1-SU1, SUN-MN									
Active Enclosure Support	Enclosure Support 0									
Active Alarm Inclusion	Alarm Inclusion 0									
Active Enclosure	Enclosure Width 280	Enclosure Height 200	Enclosure Length 480	Enclosure Material metal						
Active Terminal Block Amp	Terminal Block Amp NA									
Active Controller	DS1 0	DS2 0	DS3 1	DS4 0	DS5 0	DS6 1	DS7 0	DS8 0		
Active Pole	Pole 3 0	Pole 2 0	Pole 1 1	Pole N 1						
Active Switch	Switch Voltage 110	Switch Amperage 50	Switch width 19	Switch Height 85	Switch Length 95					
Active Terminal Block Inclusion	Terminal Block Inclusion 0									
Active Neutral	Neutral Grounded 1									
Active Load	Load Amperage 40	Load Wires 2								
Active Generator	Waiting Time 2	Monitoring 0								
Active Utility	Utility Voltage 110	Utility Frequency 60	Utility Phases 1							

A
3

Legend
☐ Input variable
☐ Output variable
☐ Function enabled
☐ Function disabled

Figure II-22. The solution for the customer requirements

After the customer requirements have been entered, a solution to the CN-F model is found in only one iteration of the program shown in Figure II-15. The values of the solution found are shown in Figure II-22. Note that, at this point all the variables of the type *Active* “X” have become disabled (buttons of the corresponding switches turned to the left), indicating that all of them have been executed and the corresponding output values defined. The values for the input variables, defined by the customer, are shown in the lower fields (coloured in yellow) and the values for the output variables, defined by the *d*-functions, are shown in the upper fields enclosed by (blue) frames. Note that, the variable *Terminal block inclusion* is both an input and output variable.